RÜGHEIMER
SPANIK

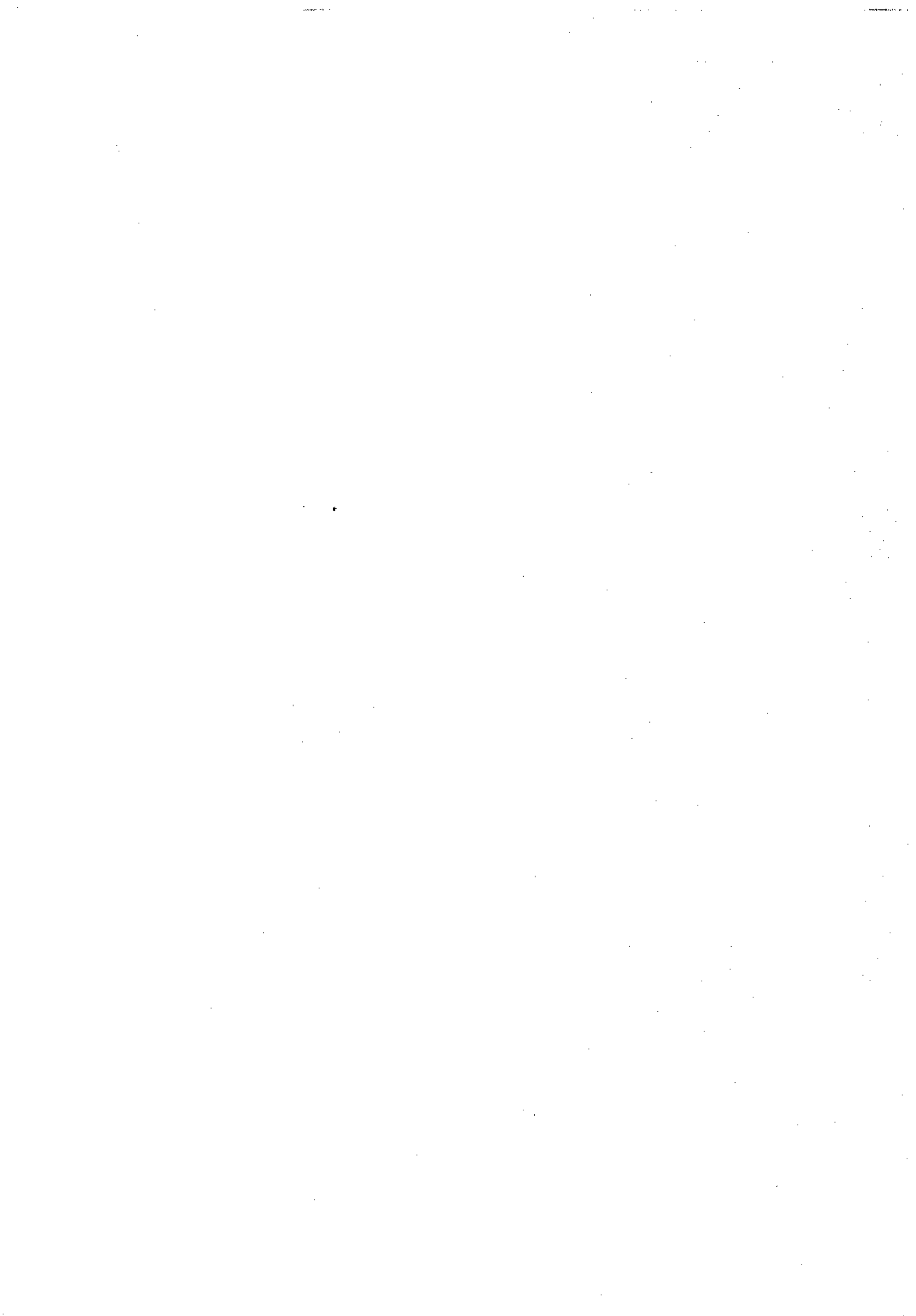
8. Auflage
Diskette im Buch

```
LIST
DEFINT P-Z
DIM P(2500)
CLS
LINE(0,0)-(120,120),,BF
ASPECT = .1
    WHILE ASPECT<20
        CIRCLE(60,60),55,0,,A
        ASPECT = ASPECT*1.4
    WEND
GET (0,0)-(127,127),P
CheckMouse:
   IF MOUSE(0)=0 THEN CheckM
   IF ABS(X-MOUSE(1)) > 2 T
   IF ABS(Y-MOUSE(2)) < 3 T
MovePicture:
   PUT(X,Y),P
   X=MOUSE(1): Y=MOUSE(2)
   PUT(X,Y),P
   GOTO CheckMouse
```

Picture

Amiga BASIC

DATA BECKER



Rügheimer
Spanik

AmigaBASIC

DATA BECKER

8. unveränderte Auflage 1991

Umschlaggestaltung Werner Leinhos

Text verarbeitet mit Word 3.01, Microsoft

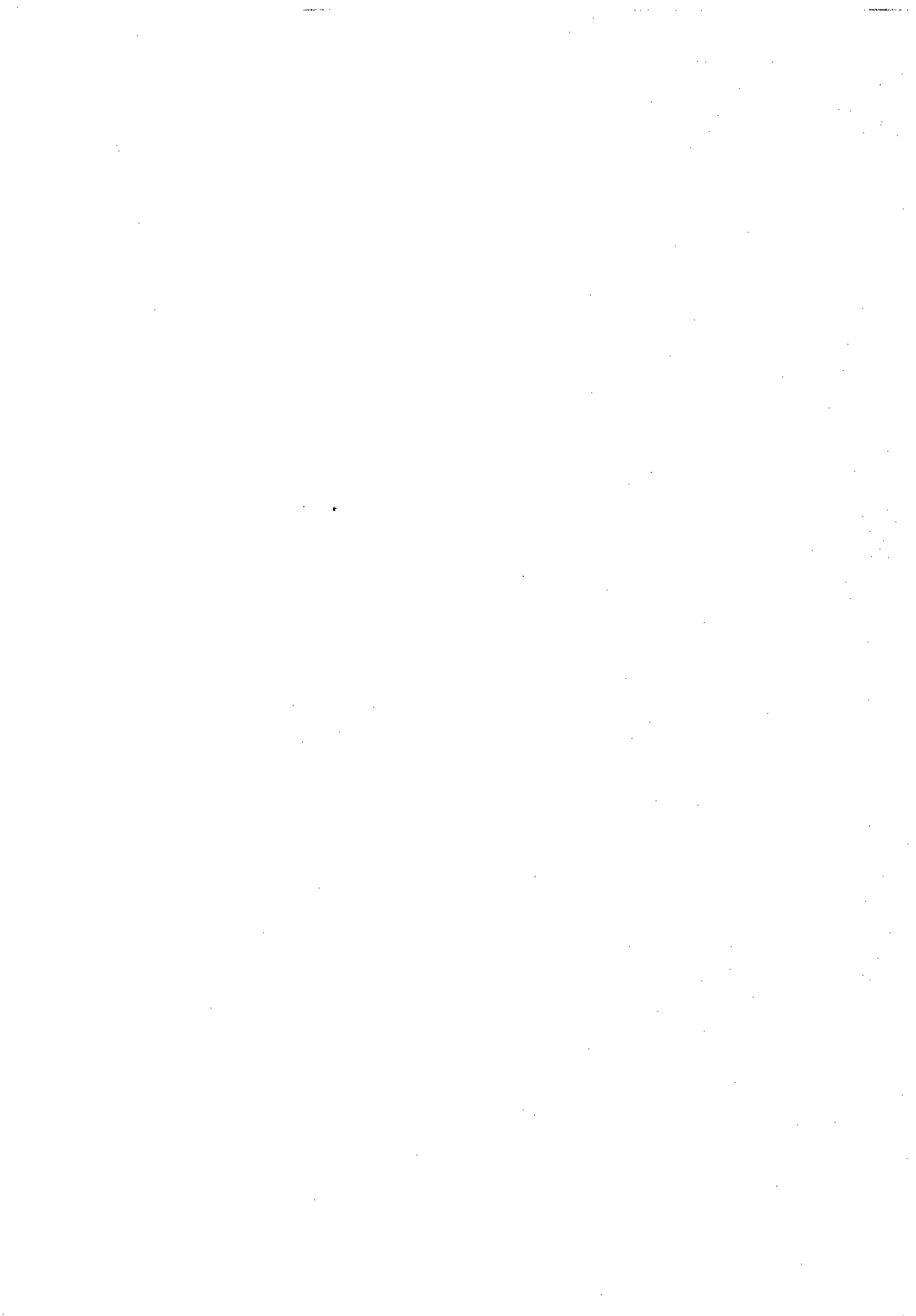
Druck und buchbinderische Verarbeitung Graf und Pflügge, Düsseldorf

ISBN 3-89011-202-9

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle technischen Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.



Für Jay Minor und all die anderen, die diese wunderbare Maschine möglich gemacht haben.

Gestatten, Rügheimer & Spanik



Christian Spanik geht unruhig auf der ersten Seite hin und her. Hannes Rügheimer steht im Hintergrund und spielt mit ein paar Disketten. Plötzlich wird es hell...

"He Hannes, da hat einer das Buch aufgeschlagen."

"Dann zieh' doch mal deine Fliege gerade und sag' Guten Tag."

Christian fummelt an der Fliege herum. Dann beide im Chor: "Guten Tag."

Christian (etwas verlegen): "Ja... ähm... Nett, daß Sie bei uns vorbeischauen auf der Suche nach einem BASIC-Buch zu Ihrem Amiga."

Hannes (drängelt sich nach vorne): "Jetzt wollen Sie natürlich wissen, was Sie davon haben, wenn Sie dieses Buch nehmen und nicht das andere da..."

Christian rempelt Hannes an, dann leise: "Pssssst."

Christian (zum Leser): "Kümmern Sie sich gar nicht um andere Bücher. Wir wollen Ihnen jetzt lieber mal einen kurzen Überblick geben, was Sie hier alles finden."

"Und was Sie davon haben, wenn Sie es lesen. Richtig: Sie können dann zum Beispiel Musik mit Ihrem Amiga machen oder Bilder malen..."

"Das sieht er doch sowieso im Inhaltsverzeichnis. Da gucken die Leser doch immer zuerst..."

Christian (etwas aus dem Konzept gekommen): "Ja - was will ein Leser denn dann wissen?"

"Na die tollen Dinger, die Highlights."

"Ach du meinst damals, als wir zusammen in diese Diskothek...?"

"Nein, die im Buch natürlich. Laß mich mal. Wissen Sie, wenn's ums Programmieren geht, kann man mit dem Christian wirklich nicht viel anfangen. Schreiben ja, aber programmieren - das muß ich Ihnen schon erklären. Also, im Prinzip finden Sie in diesem Buch alles, was man zum Amiga wissen muß, wenn man ihn in BASIC programmieren will. Klar, das ist eine ganze Menge, aber danach wissen Sie auch wirklich Bescheid."

Christian (mischt sich ein): "Moment, noch etwas Wichtiges: Weil es soviel ist, was man da verstehen muß, und weil program-

mieren nicht immer ganz einfach ist, haben wir versucht, alles so zu erklären, daß Sie es auch wirklich verstehen. Nebenbei soll Ihnen das Lernen aber auch Spaß machen..."

Hannes nickt bekräftigend, steckt die Disketten in die Tasche: "Richtig. Das ist auch der Grund, warum wir das Buch zusammen gemacht haben. Denn wir haben entdeckt, daß wir ganz gut zusammen schreiben können. Wissen Sie, es ist nämlich nicht unser erstes Buch. Aber das ist eine andere Geschichte..."

Christian (in leichter Hektik, weil er Angst hat, der Leser könnte weiterblättern): "Wenn Sie sich nachher mal das Inhaltsverzeichnis anschauen, werden Sie sehen, was da alles drin ist. Egal, ob Sie die Grafikmöglichkeiten des Amiga oder seine musikalischen Fähigkeiten ausnutzen wollen. Alles was dazu wichtig ist, finden Sie hier. So beschrieben, daß Sie es verstehen und später in der Lage sind, eigene Programme auf dem Amiga zu schreiben."

Hannes (blättert wild in einem dicken Packen Manuskriptpapier): "Mal sehen... Natürlich bietet das Buch auch ein paar Bonbons: Zum Beispiel haben wir einige Nächte damit verbracht, bis wir herausbekommen hatten, wie Sie Bilder aus Graphicraft oder anderen Malprogrammen in BASIC verwenden können..."

Christian (unterbricht Hannes): "Und umgedreht: Sie können jetzt alle Bilder, die Sie in BASIC erzeugen, im sogenannten IFF-Format abspeichern. Wenn Ihnen das jetzt nichts sagt, macht nichts. Zum Erklären sind wir ja da. Sie brauchen also nichts weiter mitzubringen als ein wenig Lust, die nächsten paar hundert Seiten mit uns zusammen am Amiga zu verbringen. Wir versprechen Ihnen, daß Sie dann eine ganze Menge lernen können und auch der Spaß nicht zu kurz kommt. - Au, verdammt."

"Was ist denn?"

"Mensch, wir haben noch gar kein Vorwort! Wir sprechen hier einfach Leser an und haben noch nicht einmal ein Vorwort!"

"Stimmt ja, dabei dachte ich, wir sind fertig. Wie machen wir das bloß..."

Beide ziehen sich gedankenverloren zurück.

Vorhang.

Vom Umgang mit Büchern - besonders mit diesem

Unser AmigaBASIC-Buch richtet sich sowohl an Anfänger als auch an Fortgeschrittene. Je nach Ihrem Kenntnisstand können Sie sich eine von zwei Einleitungen aussuchen. Doch dazu erfahren Sie gleich mehr in Kapitel 0.

Die Kapitel danach sind dann für alle Leser gedacht. Wer schon viel von Computern versteht, wird wahrscheinlich das eine oder andere bekannte finden. Trotzdem sollten auch die Fortgeschrittenen und Profis ein Kapitel nach dem anderen lesen und nichts auslassen. Es gibt nämlich viele Hinweise auf Dinge, die beim Amiga ein bißchen anders sind als bei anderen Computern.

Wer noch keine Computer-Erfahrungen hat, für den gilt natürlich erst recht: Bitte lesen Sie alles, und springen Sie nicht zwischen den Kapiteln. Wir werden Stück für Stück in die Welt von AmigaBASIC vordringen. In den späteren Kapiteln bauen wir natürlich auf das auf, was Sie vorher gelernt haben.

Im Text stehen *kursiv gedruckte Worte*. Diese Worte sind Fachausdrücke, deren Erklärung im Text zu weit führen würde. Alle Fachworte finden Sie mit Erklärung im Anhang D, unserem kleinen Fachwortlexikon. Da wir gerade von den Anhängen sprechen: Dort gibt es noch einiges mehr zu finden.

Bei den ersten Versuchen machen Sie zwangsläufig Fehler. AmigaBASIC bringt dann eine Fehlermeldung. Meist gehen diese Fehler auf Tippfehler oder eine falsche Reihenfolge von Werten zurück. Im Text weisen wir auf typische oder mögliche Fehler hin. Alle Fehler können wir aber nicht vorhersehen, je nach der Vorgeschichte Ihres Programms und den internen Zuständen im Amiga kann es da sehr viele Möglichkeiten geben. Das sorgt für Abwechslung und für den Anhang A. Dort finden Sie alle Fehlermeldungen in alphabetischer Reihenfolge. Wenn Sie hinten nachschauen, finden Sie Erklärungen, warum der Fehler auf-

getreten sein kann, was er bedeutet, und was man dagegen unternehmen kann. Schlagen Sie also im Falle eines Falles im Anhang A nach.

Apropos abtippen: Im Buch finden Sie eine ganze Menge Beispielprogramme. Das reicht von kleinen Dreizeilern bis hin zu mehreren Seiten langen Listings. Das Beste wäre natürlich, wenn Sie alle diese Listings gemeinsam mit uns eingeben würden. Im Verlauf der einzelnen Kapitel stellen wir Ihnen immer wieder einen neuen Programmabschnitt vor, erklären die neuen Befehle und wie die einzelnen Teile funktionieren. So läßt sich unserer Meinung nach der größte Lerneffekt erreichen.

Uns ist aber auch die Schattenseite dieses Verfahrens bewußt. Viele Leser haben einfach nicht die Zeit - vielleicht auch nicht die Lust?!-, seitenlange Listings abzutippen. Von den vielen Fehlern, die man dabei zwangsläufig macht, gar nicht zu reden. Und daß das Abtippen auch keinen so großen Spaß macht, braucht uns auch keiner zu sagen.

Also haben wir uns entschlossen, diesem Buch eine Diskette beizufügen. Auf dieser "Diskette im Buch" finden Sie alle Programme, die wir in den verschiedenen Kapiteln vorstellen. Einfaches Anklicken genügt, dann können Sie jedes einzelne Programm sofort ausprobieren. Das einzige, was Sie vorher noch tun müssen, ist, das AmigaBASIC von Ihrer "Extras"-Diskette auf unsere Diskette zu kopieren. Wir kommen darauf nochmal im "Zwischenspiel 1" zurück.

In diesem Zusammenhang sollten wir aber vielleicht einem Mißverständnis vorbeugen: Die Programme in diesem Buch und auf der Diskette haben allein das Ziel, Ihnen beim Lernen von AmigaBASIC zu helfen. Wir wollen und können nicht mit professioneller Software konkurrieren. Daß Programme wie "Aegis Videotitler", "Deluxe Paint II" oder "Analyze" deutlich mehr leisten als unser "Videotitel-Programm", unser Malprogramm oder das "Balken- und Tortengrafik-Utility", das versteht sich eigentlich von selbst.

Mit der Diskette im Buch ist all denen geholfen, die sich unsere Beispiele erst mal komplett anschauen wollen. Sie können ja zu einem späteren Zeitpunkt das Listing auf dem Bildschirm verfolgen und es mit unseren Erklärungen vergleichen.

Wenn Sie ein Programm selbständig eingegeben haben und dieses Programm dann absolut nicht laufen will, weil irgendwo ein Fehler steckt - auch dann hilft Ihnen die Diskette im Buch. Sie können in so einem Fall das Programm laden und ausprobieren. Wenn Ihr Amiga soviel Speicher hat, daß Sie AmigaBASIC zweimal starten können (das ist etwa ab 1 MegaByte sinnvoll), könnten Sie auch gleichzeitig Ihre und unsere Version laden, die beiden List-Windows nebeneinander legen und Zeile für Zeile vergleichen. Bis Sie den Fehler schließlich entdeckt haben.

Kommen wir von unserer Diskette zu einer anderen. Die Firma Commodore hat Ihnen einige Beispielprogramme auf Ihrer "Extras"-Diskette mitgeliefert. Im Anhang C erklären wir Ihnen kurz, wie sie funktionieren, was Sie beachten müssen und was Sie damit anfangen können.

Irgendwann haben Sie unser Buch fertig gelesen und werden eigene BASIC-Programme schreiben. Dabei treten sicher Fragen auf über die Verwendung eines BASIC-Befehles und seine Möglichkeiten. Damit Sie nicht lange im Text des vorderen Buchteils suchen müssen, gibt es hinten den Anhang B, in dem alle BASIC-Befehle mit allen Optionen (Möglichkeiten) erklärt sind. Da wir zum Programmieren nicht alle Befehle und alle Optionen benutzen, ist dieser Anhang schon während des Lesens nützlich und interessant.

In diesem Buch wird manchmal die Rede sein von Amiga 500, 1000 und 2000. Wie Sie vielleicht wissen, hatte der Amiga schon vom ersten Jahr an eine bewegte Geschichte. Der Ursprungs-Amiga, der ganz zu Beginn zu kaufen war, hat die Typenbezeichnung Amiga 1000. Im Frühjahr 1987 wurden seine beiden Nachfolger vorgestellt: Der Amiga 500, ein Kompaktgerät, in dem Tastatur, Floppy und Rechner in einem einzigen Gehäuse untergebracht sind, und der Amiga 2000, ein offenes System, in dem Sie Erweiterungskarten einstecken können und das

vom Design her eher einem PC ähnelt. Der Amiga 1000 wird seitdem nicht mehr hergestellt. Nichtsdestotrotz gibt es, abgesehen von unseren beiden, eine ganze Menge Amiga 1000, die in Deutschland bereits verkauft sind. Deshalb haben wir unser Buch so geschrieben, daß es von allen Amiga-Besitzern gleichermaßen genutzt werden kann. Das war auch nicht allzu schwer, da die drei Amiga-Brüder untereinander sowieso voll kompatibel sind. An einigen Stellen finden Sie Anmerkungen, was sich bei der Handhabung der einzelnen Amiga-Versionen unterscheidet.

Hier haben wir gleich den ersten Hinweis. Er richtet sich nur an die Besitzer von Amiga 1000, die keine Speichererweiterung haben und mit 256 KByte RAM (Schreib-/Lese-Speicher) auskommen müssen. Falls Sie einen Amiga 500 oder 2000 haben, ist dieses Thema für Sie uninteressant. Wir gehen nämlich bei den Programmbeispielen in diesem Buch davon aus, daß mindestens 512 KByte RAM zur Verfügung stehen. Für die Besitzer von nicht aufgerüsteteten 1000er-Amigas haben wir nur einen Tip: Gehen Sie so schnell wie möglich in Ihren Laden, und kaufen Sie sich die 256 KByte-Erweiterung für Ihren Amiga. Denn erstens weiß man nicht, wielange dieses Zusatzteil überhaupt noch erhältlich ist, und zweitens ist die Speichererweiterung auf 512 KByte das sinnvollste Zusatzteil überhaupt. Ohne sie brauchen Sie gar nicht erst an die Anschaffung eines zweiten Diskettenlaufwerks oder anderer Peripheriegeräte zu denken...

Aber, wie gesagt, dieser Hinweis gilt ausschließlich für die Besitzer von Amiga 1000 ohne 256KByte-Speichererweiterung. Und das dürften nicht mehr allzuviele unter Ihnen sein.

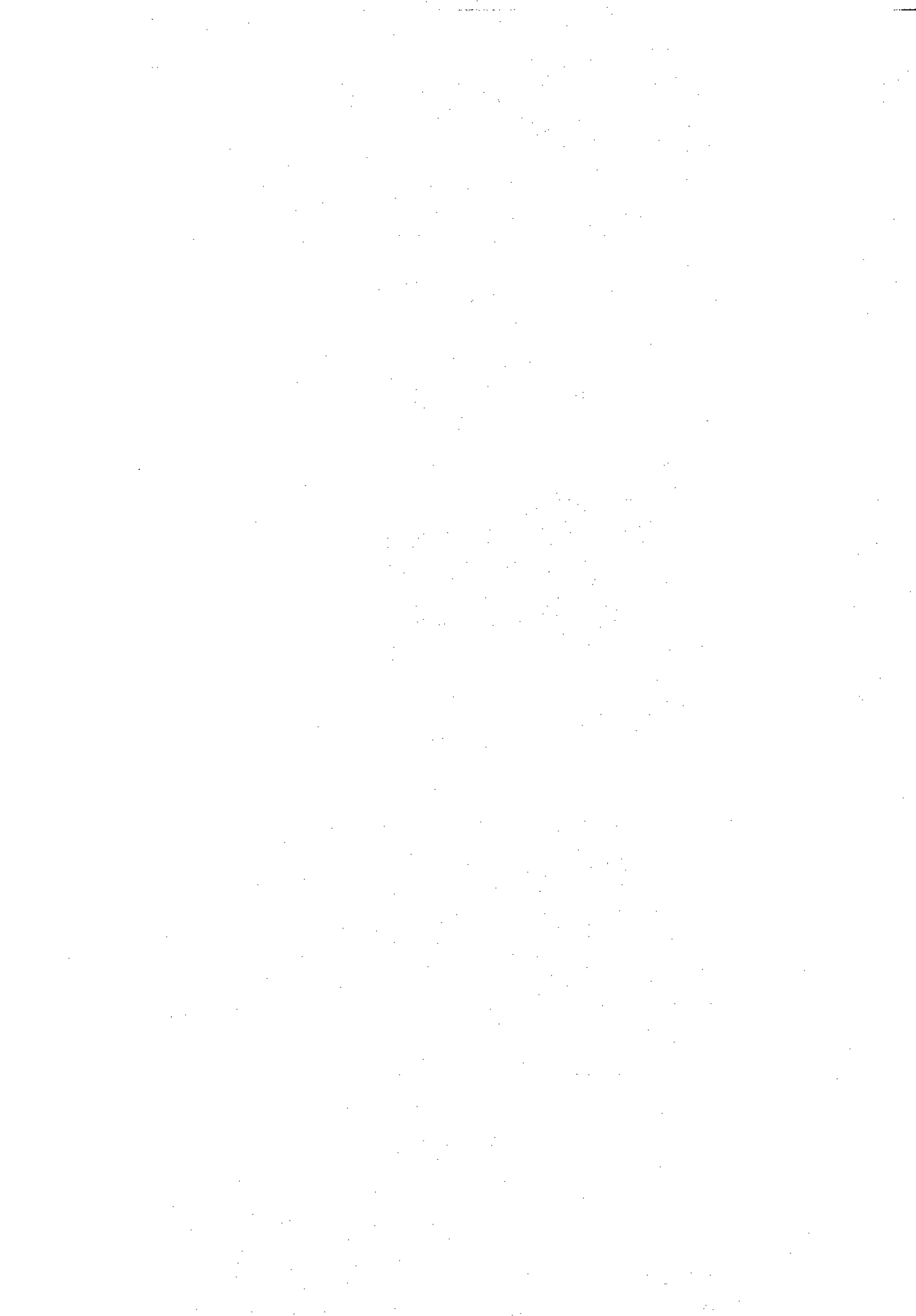
Und jetzt noch ein paar organisatorische Punkte: Im Buch gehen wir davon aus, daß Sie die Grundfarben der Original-Workbench benutzen. Darauf beziehen sich alle Farbangaben. Außerdem sind alle Programme für die Darstellung mit 80 Zeichen formatiert. Das heißt: Wenn Ihre Texte bei den Programmen, die Sie hier abtippen, richtig auf dem Bildschirm ausgegeben werden sollen, müssen Sie den Amiga mittels Preferences auf 80-Zeichen-Darstellung schalten. Wie das geht, steht im Amiga-Benutzerhandbuch.

Außerdem gehen wir in diesem Buch davon aus, daß Sie mit den 1.2-Versionen der Disketten "Kickstart" (nur für Besitzer des Amiga 1000), "Workbench" und "Extras" arbeiten. Sollten Sie diese Disketten noch nicht besitzen, wenden Sie sich bitte an Ihren Händler, der sie Ihnen sicher kopieren wird. Sollte das aus irgendwelchen Gründen nicht gehen, versuchen Sie, die Disketten vielleicht von befreundeten Amiga-Besitzern zu erhalten. Wer einen Amiga 500 oder 2000 hat, bekommt die richtigen Disketten sowieso mitgeliefert.

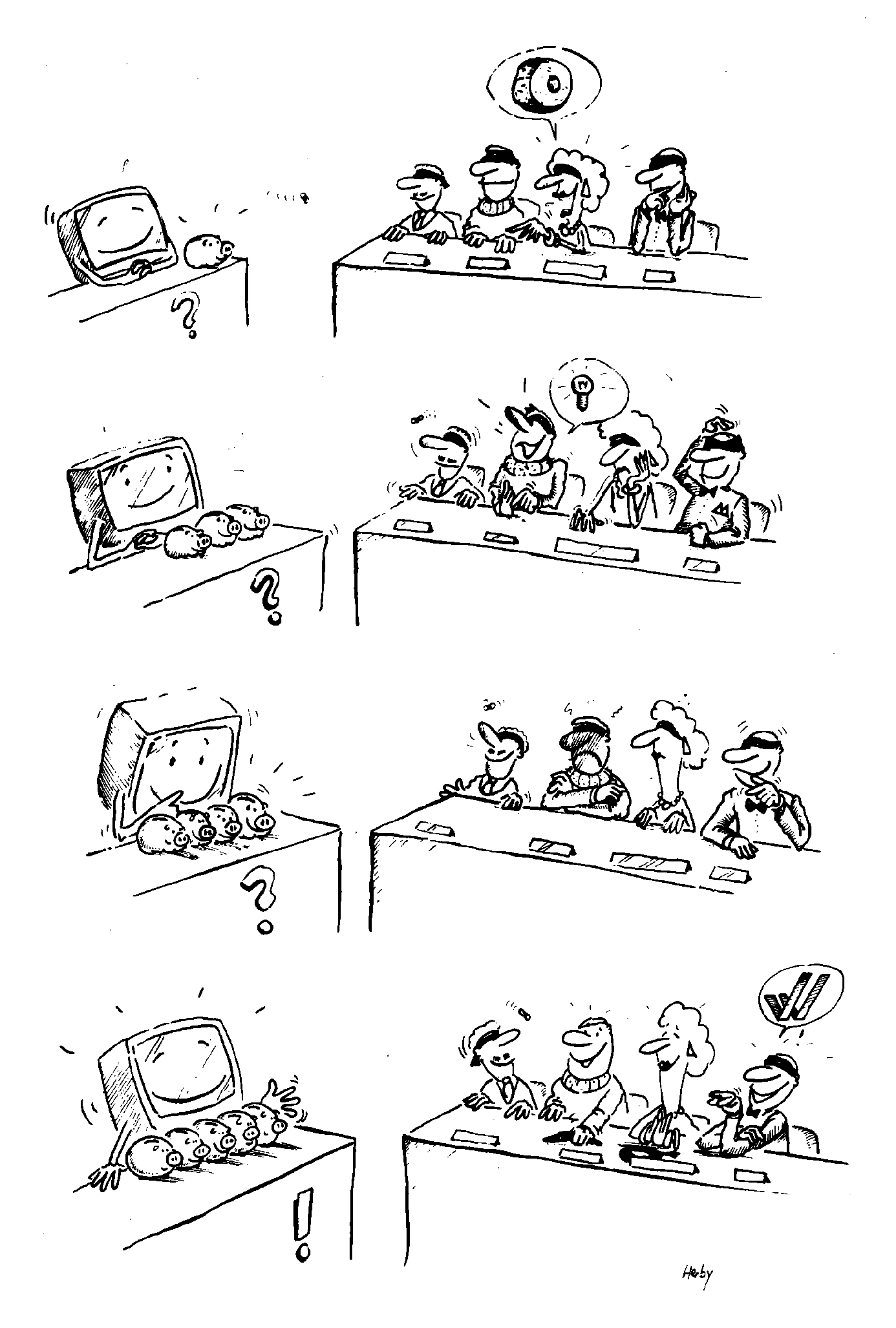
Apropos "Extras"-Diskette: Commodore hat in verschiedenen Lieferungen dieser Diskette verschiedene Namen gegeben. Die häufigsten Varianten sind: "Extras", "ExtrasD" oder "Extras 1.2". Für die Beispiele in unserem Buch ist es wichtig, daß wir alle vom selben Diskettennamen ausgehen. Falls Ihre "ExtrasD"-Diskette anders heißt, benennen Sie sie bitte in "ExtrasD" um. Sollten Sie nicht wissen, wie das geht, so erfahren Sie es im Commodore-Anwender-Handbuch, Kapitel 4.4.6.

So. Aber jetzt kann's endlich losgehen.

Inhaltsverzeichnis



Kapitel 0: Was bin ich?



Im folgenden kleinen Abschnitt sollen Sie nicht heiteres Berufe-raten mit uns spielen, sondern für sich herausfinden, welcher Typ von Leser Sie sind. Das ist wichtig, weil unser Buch dem Anfänger genauso helfen soll wie dem Profi. Das stellt uns aber vor die Frage, welche Grundkenntnisse wir eigentlich voraussetzen können. Kennen Sie sich mit der Bedienung des Amiga schon perfekt aus? Welche Erfahrungen haben Sie überhaupt mit Computern?

Um Ihnen einen optimalen Einstieg zu bieten, haben wir uns entschlossen, einfach zwei verschiedene Einleitungskapitel zu schreiben: Eines für den, der bisher noch nicht viel mit Computern zu tun hatte und auch den Amiga noch nicht lange besitzt. Ein anderes für den, der schon einige Computererfahrungen hat.

Sie können sich jetzt selbst aussuchen, mit welchem Kapitel Sie lieber anfangen möchten. Diejenigen, die sich zuerst noch mal alles erklären lassen wollen, bitten wir ins Kapitel 0.1. Die anderen Leser werden gebeten, sich im Kapitel 0.2 einzufinden.

In jedem Fall ist die Programmiersprache AmigaBASIC für alle geeignet, die eigene Programme schreiben wollen, unabhängig von den Vorkenntnissen. Die Entscheidung, die Sie jetzt für sich treffen sollten, hieße auf BASIC übrigens:

```
IF NOT Profi THEN GOTO Kapitel 0.1
IF Profi THEN GOTO Kapitel 0.2
```

Das sollten Sie allerdings nicht Ihren Amiga entscheiden lassen. Er würde bei diesen Zeilen als Ergebnis höchstens einen SYNTAX ERROR produzieren. Warum? Die Antwort auf dieses und andere Geheimnisse werden Sie auf den nächsten Seiten finden. Aber sehen Sie sich vor: Nach diesem Buch werden Sie in AmigaBASIC kaum mehr Geheimnisse entdecken, aber dafür eine ganze Menge Spaß.

0.1 Von Mäusen, die fensterln - die Workbench

Wenn Sie sich für dieses Kapitel entschieden haben, sind Sie der Meinung, daß Sie sich noch nicht so gut mit der Workbench und den Grundfunktionen des Amiga auskennen. Oder daß zumindest ein wenig Auffrischung nicht schaden könnte. Wir wollen diese Themen deshalb besonders ausführlich besprechen.

- Schalten Sie dazu bitte den Amiga und den Monitor ein.

Auf dem Monitor sehen Sie zunächst nichts weiter als eine einfarbige, dunkelgraue Fläche. Aber keine Sorge, das heißt nicht, daß man beim Arbeiten mit Amiga in grauer Trostlosigkeit versinkt: Der Amiga führt nur einen Selbsttest durch. Deshalb hören Sie auch gleich danach eine kurze Tonfolge. Sie kommt entweder aus dem Lautsprecher des Monitors oder über Ihre Stereoanlage, je nachdem, wie Sie den Amiga angeschlossen haben. Schließlich wird der Bildschirm weiß, dann sehen Sie eine stilisierte Hand, die eine Diskette hält. Womit Sie bereits das erstemal mit einem wichtigen Feature des Amiga konfrontiert werden, der Symbol-Sprache. In diesem Fall bedeutet das Symbol: Ihr Amiga verlangt eine Diskette. Im Falle des Amiga 1000 ist es die Kickstart-Diskette. Diese Diskette ist für ihn so wichtig wie für manche Leute der Morgenkaffee - er braucht sie einfach, um in Gang zu kommen.

- Legen Sie bitte die Kickstart-Diskette ein, wenn Sie einen Amiga 1000 besitzen.

Schieben Sie die Diskette dazu bitte so in das interne Laufwerk des Amiga, wie auf dem Bildschirm gezeigt: Der Metallverschluß der Diskette muß nach vorn zeigen und die beschriftete Seite nach oben. Auf der Kickstart-Diskette befindet sich das *Betriebssystem* des Computers. Ein paar Worte zur Klärung des Begriffes: Selbst der tollste Computer steht ziemlich dumm da, wenn er kein Programm hat, das ihm sagt, was er tun soll. Das *Betriebssystem* ist ein solches Programm. Hier erfährt Ihr Amiga zum Beispiel, wie er auf die Tastatur und auf die Maus reagieren oder wie er Grafiken und Texte auf den Bildschirm bringen soll. Diese Informationen werden während des Ladens der

Kickstart in einen speziellen Speicherbereich geschrieben, der später nicht mehr gelöscht werden kann, solange der Amiga angeschaltet bleibt. Nach etwa 20 Sekunden ist er mit dem Laden fertig. Nun verlangt er nach einer neuen Diskette - wieder durch das bekannte Handsymbol. Die Diskette trägt diesmal die Aufschrift "Workbench".

- Drücken Sie bitte auf den Auswurf-Knopf am Diskettenlaufwerk.

Die alte Diskette wird ausgeworfen, das heißt, sie schaut dann ein Stück aus dem Laufwerk heraus.

- Entnehmen Sie bitte die Kickstart-Diskette.

- Schieben Sie dann die Workbench-Diskette ins interne Laufwerk.

Und an dieser Stelle begrüßen wir auch diejenigen, die einen Amiga 500 oder 2000 haben. Ihr Computer hat das Betriebssystem fest eingebaut, es muß nicht erst von Diskette geladen werden. Für Sie beginnt die Einschaltprozedur gleich mit der Workbench.

Wieder beginnt das Laufwerk zu surren. Nach kurzer Zeit ist auch die Workbench-Diskette geladen. Sie sehen dann einen blauen Bildschirm mit einer weißen Kopfzeile und auf der rechten Seite ein Symbol mit dem Namen "Workbench 1.2" darunter. Das Symbol steht für die Diskette im Laufwerk. Wann immer Sie eine neue Diskette in ein angeschlossenes Laufwerk einlegen, erscheint ein solches Symbol auf der Workbench-Oberfläche. Solche Symbole - egal wofür sie stehen - heißen Icons (engl.: Abbilder, Ikonen. Hier eben am besten mit "Symbol" übersetzt). Frei nach dem Motto "Der Amiga-Besitzer läßt das Mausen nicht" geht's jetzt endlich los.

- Bewegen Sie bitte die Maus auf der Tischplatte.

Wenn Sie die Maus hin- und herschieben, sehen Sie, wie sich der rote Pfeil auf dem Bildschirm synchron zur Maus bewegt. Aus diesem Grund heißt der Pfeil auch Mauscursor oder Mauszeiger.

- Bewegen Sie den Pfeil auf das Icon der Workbench-Diskette.

- Drücken Sie dann einmal auf die linke Taste der Maus.

Sie sehen, wie die Diskette auf dem Bildschirm schwarz wird. Warum? Ganz einfach: Das Anklicken mit der linken Maustaste aktiviert ein Symbol auf der Workbench-Oberfläche. Damit der Amiga eine Möglichkeit hat, Ihnen zu zeigen, welches Objekt gerade aktiv ist, verändert er die Farbe. Wenn Sie außerhalb des Symbols irgendwo auf den Workbench-Bildschirm klicken, wird die Diskette wieder weiß. So können Sie Symbole deaktivieren. Normalerweise kann immer nur ein Symbol aktiv sein.

- Drücken Sie nun bitte zweimal kurz hintereinander die linke Maustaste.

Sie werden gleich darauf Zeuge einiger Betriebsamkeit. Der Mauscursor hat sich in ein Wölkchen verwandelt, das den Eindruck erweckt, als wollte es jeden Moment davonschweben. Auch das Wölkchen ist symbolisch gemeint: Der Amiga teilt uns mit, daß er vorerst beschäftigt ist. Jetzt müssen Sie kurz warten. Es entsteht ein Rahmen auf dem Bildschirm und nacheinander erscheinen verschiedene Icons innerhalb dieses Rahmens. Am Ende der ganzen Aktion bietet sich Ihnen folgendes Bild:

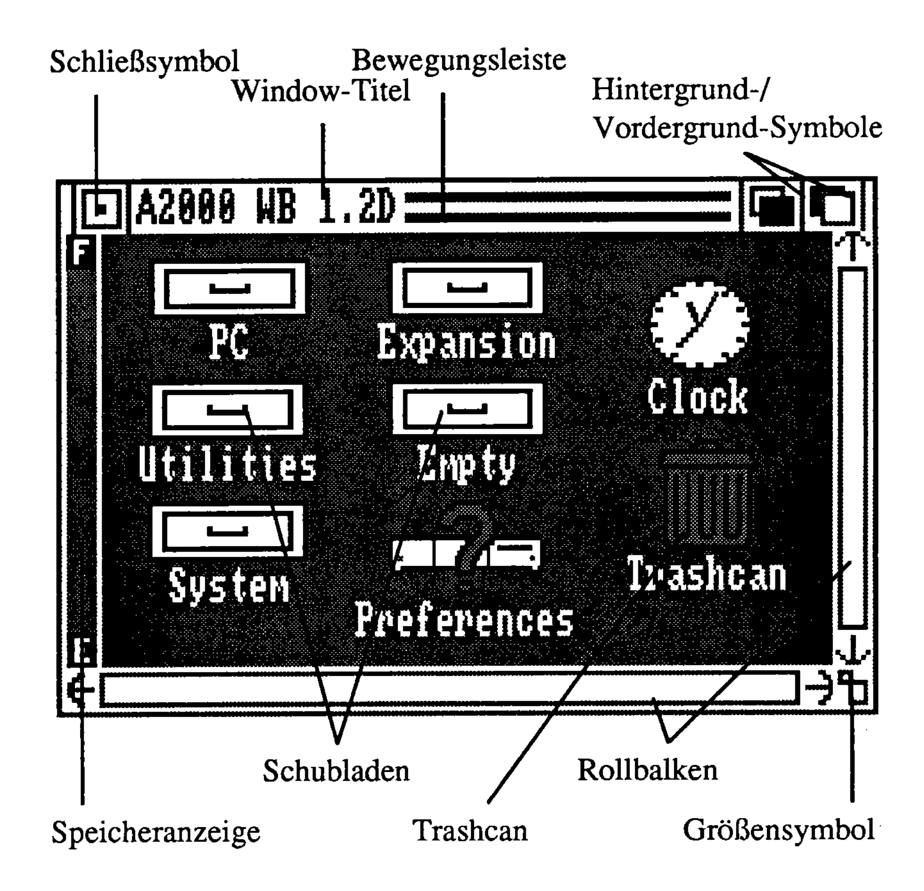


Bild 1: **Das Workbench-Window**

Das Ding im Bild ist ein Window (auf Deutsch also: Fenster). Solche Windows dienen dazu, den Inhalt von Disketten darzustellen. Im Fensterrahmen, also der Umrandung des Windows, befinden sich einige wichtige Dinge. Was es da im einzelnen gibt, sehen Sie auf dem Bild.

Window-Titel

Jedes Window hat einen Namen. In diesem Fall "Workbench 1.2". Wenn dieser Name etwas unleserlich aussieht, liegt das nicht an Amigas unleserlicher Handschrift, sondern bedeutet, daß das Window nicht aktiviert ist. Bewegen Sie in diesem Fall den Mauscursor in das Window und klicken Sie einmal mit der linken Maustaste: Nun ist das Window angewählt, die Schrift im Titel ist gut zu lesen.

Bewegungsleiste

Wenn Sie den Mauscursor auf dieses Feld bringen, die linke Maustaste gedrückt halten und dann die Maus bewegen, können Sie das ganze Window auf dem Bildschirm hin- und herschieben. Dabei wird zunächst nur ein oranger Rahmen in der Größe des Windows bewegt. Daran können Sie erkennen, wie das Fenster liegen würde und was es alles überdecken würde, wenn Sie die Maustaste wieder loslassen. Erst dann wird das neue Window vollständig abgebildet.

Hintergrund-/Vordergrund-Symbole

Wenn sich mehrere Windows überlagern, können Sie mit diesen Symbolen auswählen, welches Window sich im Vordergrund befinden soll. Um das auszuprobieren, wählen Sie bitte das Icon "Clock" an. (Wenn Sie nicht mehr genau wissen, wie das geht, hier nochmals zum Spicken: Bewegen Sie den Mauscursor auf das Symbol der Uhr (=engl. Clock) und klicken Sie zweimal drauf.)

Nach wenigen Sekunden erscheint eine Uhr auf dem Bildschirm. Diese Uhr wird zwar wahrscheinlich eine falsche Uhrzeit anzeigen, aber das soll uns im Moment nicht stören. (Wenn es Sie doch stört: Mit dem Programm "Preferences" können Sie unter anderem auch die Uhr stellen. Wie "Preferences" bedient wird, ist im Amiga-Anwender-Handbuch in Kapitel 5.1 beschrieben. Wenn Sie dort jetzt nachgucken wollen, vergessen Sie nicht, sich

ein kleines Zettelchen hier zwischen die Seiten zu legen, damit wir uns wiederfinden, wenn Sie das erledigt haben!) Die Uhr muß teilweise das Workbench-Window überlagern. Wenn sie das nicht tut, verschieben Sie sie mit der Bewegungsleiste bitte so, daß sie es doch tut. Danke.

- Bringen Sie den Mauscursor auf das rechte der beiden Symbole am Fensterrahmen des Workbench-Windows und klicken Sie einmal.

Die Uhr verschwindet hinter dem Window.

- Klicken Sie bitte ins linke der beiden Symbole.

Schon ist die Uhr wieder im Vordergrund.

Die kleinen Kästchen innerhalb der Symbole für Vordergrund und Hintergrund sollen Windows darstellen. Das weiße Kästchen steht dabei für das Window, an dessen Rahmen sich die beiden Symbole befinden. In unserem Beispiel also das Uhr-Window. Das schwarze Kästchen vertritt ein oder mehrere andere Windows, die dahinter liegen, was bei uns im Moment aber nur dem Workbench-Window entspricht. Das zweite Symbol - weißes Kästchen hinter schwarzem Kästchen - ist das Hintergrundsymbol. Klickt man es an, wird das zugehörige Window in den Hintergrund gelegt.

Größensymbol

Wenn Sie dieses Symbol mit dem Mauscursor festhalten (sprich: Mauscursor darauf positionieren und linke Taste gedrückt halten), können Sie die Größe des Windows verändern. Wie beim Window-Verschieben zeigt auch hier wieder ein oranger Rahmen die Umrisse des neuen Windows an. Erst nach dem Loslassen der Maustaste wird das Window vollständig auf den Bildschirm gebracht.

Rollbalken

Wenn ein Window so klein ist, daß der gesamte Inhalt nicht mehr auf einmal dargestellt werden kann, zeigt das dieser Balken an. Sie erkennen es daran, daß oben oder unten blaue Flächen entstehen. An denen können Sie auch ungefähr abschätzen, welcher Teil des Inhalts gerade sichtbar ist. Angenommen ober- und unterhalb des Balkens ist jeweils ein Drittel Blau zu sehen, dann heißt das, daß im Moment nur das mittlere Drittel innerhalb des Fensters dargestellt werden kann. Durch Verschieben dieses Balkens kann ein bestimmter Teil ausgewählt werden. Anstatt den Balken zu verschieben, können Sie auch einen der Pfeile anklicken, die Sie über und unter dem Balken sehen. Auch dann wird der Windowinhalt in die angegebene Richtung verschoben - allerdings nur ein kleines Stück. Das Ganze funktioniert sowohl in vertikaler als auch in horizontaler Richtung. Natürlich können Sie auch einfach das Window vergrößern, wie vorher beschrieben.

Speicheranzeige

Dieser Balken gibt bei Diskettenwindows Auskunft darüber, wieviel Prozent der Diskette schon belegt sind. E steht für Empty (Leer), F für Full (Voll). Je höher der Balken, desto voller ist die Diskette.

Schließsymbol

Wenn Sie in dieses Symbol klicken, schließt sich das Window wieder. Das heißt, es verschwindet vom Schirm.

Schubladen

Diese Schubladen sind Unter-Inhaltsverzeichnisse. Wenn man sie anwählt, erscheint ein neues Window mit weiteren Programmen oder Unter-Inhaltsverzeichnissen. Sie können sich eine Diskette wie einen Schrank voller Informationen vorstellen. Damit man

die Übersicht behält und nicht alles durchwühlen muß, wenn man etwas sucht, gibt es Schubladen, in denen alles Mögliche zu einem bestimmten Thema zu finden ist. Wenn Sie diese Schubladen aufmachen (bzw. anklicken), wird der Inhalt sichtbar. Sie können das gern einmal mit einer der Workbench-Schubladen ausprobieren.

- Klicken Sie bitte eine der Schubladen im Workbench-Window an.

Es öffnet sich ein neues Window, das zur angeklickten Schublade gehört. Dort befinden sich weitere Icons. Bitte schließen Sie das neu entstandene Window durch Anklicken des Schließsymbols. Sie können nun alles, was Sie wollen, in diese Schubladen legen. Auch Schubladen in Schubladen in Schubladen sind möglich. Das ganze folgt getreu dem Schreibtisch-Prinzip: Wenn man irgendwelche Dinge in der Schublade verschwinden läßt, schaut es wenigstens auf der Oberfläche nach Ordnung aus. Übrigens wird die Workbench gern als elektronischer Schreibtisch bezeichnet.

Trashcan (deutsch: Abfalleimer)

In diesen Abfalleimer können Sie Programme befördern, die Sie nicht mehr benötigen. Wieder eine Funktion, die dem täglichen Leben nachempfunden ist. Wenn ein Programm im Trashcan liegt, heißt das aber noch nicht, daß das Programm damit schon endgültig verschwunden ist. Es fehlt dazu noch eine Art Müllabfuhr, die den Eimer ausleert. Auf die kommen wir später noch. Sie können das Trashcan-Window öffnen wie eine Schublade. In diesem Window sehen Sie dann alle Dinge, die bisher weggeworfen wurden.

- Bitte klicken Sie den Trashcan an.

Wenn Sie das jetzt ausprobiert haben, müßten Sie festgestellt haben, daß der Trashcan zur Zeit leer ist. Schließen Sie das Trashcan-Window bitte wieder.

Wie aber kann man Programme und Daten endgültig loswerden? Damit kommen wir zu den Pulldown-Menüs. Weil Sie bei diesem Buch auch Spaß haben sollen, sorgen wir bei dieser Gelegenheit gleich für ein wenig Abwechslung. Statt immer nur die linke, drücken Sie diesmal die andere Maustaste:

- Drücken Sie die rechte Maus-Taste.

Sollten Sie das nicht schon vorher ausprobiert haben, werden Sie jetzt zum ersten Mal merken, daß sich im Bereich der Kopfzeile etwas verändert. Anstelle des Textes, der da vorher stand, sehen Sie nun drei Wörter:

```
Workbench    Disk    Special
```

- Bewegen Sie den Mauscursor mit gedrückter rechter Maustaste auf das Wort "Special".

In dem Moment, wo der Mauscursor das Wort erreicht, erscheint das zugehörige Pulldown-Menü. Menü klingt ja schon sehr nach Speisekarte. Und tatsächlich werden Ihnen auch in einem Computer-Menü verschiedene Punkte zur Auswahl angeboten. Hier zum Beispiel lesen Sie die Menüpunkte "Clean Up", "Last Error", "Redraw", "Snapshot" und "Version". Einige dieser Punkte sind wieder etwas unleserlich geschrieben: Das heißt, daß sie zur Zeit nicht angewählt werden können, weil sie nicht aktiv sind.

- Halten Sie die rechte Taste gedrückt und bewegen Sie den Mauscursor nach unten.

Während der Mauscursor über die Menüpunkte hinabwandert, sehen Sie, wie jeweils eine Menü-Option schwarz hinterlegt wird. Allerdings nur bei den Menüpunkten, die aktiv sind. Pulldown nennt man diese Art von Menü, weil das Ganze aussieht wie ein Rollo, das heruntergezogen wird. Die rechte Maustaste wird auch Menü-Taste genannt, weil mit ihr die Pulldown-Menüs bedient werden. Im Gegensatz dazu hat die linke Maustaste keinen eindeutigen Namen. Falls Sie mal in anderen

Büchern Ausdrücke wie "Auswahltaste", "Aktionstaste" oder "Aktivierungstaste" lesen: Damit ist die linke Taste der Maus gemeint. Jetzt aber zu den Pulldowns und was man damit machen kann:

- Verlassen Sie bitte das "Special"-Pulldown, und schließen Sie alle Windows auf dem Bildschirm.

- Aktivieren Sie dann bitte die Workbench-Diskette. (Also nur einmal klicken.)

- Wählen Sie nun bitte aus dem "Workbench"-Pulldown die Funktion "Open".

Noch mal schrittweise, was das bedeutet: Drücken Sie die rechte Maustaste, halten Sie sie gedrückt, bewegen Sie den Mauscursor auf das Wort "Workbench" und ziehen Sie ihn dann im Pulldown-Menü auf das Wort "Open". Wenn "Open" schwarz hinterlegt ist, lassen Sie die rechte Maustaste los. Sie sehen, daß nun dasselbe passiert, was Sie auch durch den Doppelklick erreicht haben: Das Workbench-Window wird geöffnet. Jetzt kennen Sie schon zwei Methoden, ein Disketten-Icon zu öffnen.

Sollte die Workbench-Diskette nicht mehr schwarz sein, aktivieren Sie sie bitte wieder.

- Wählen Sie dann bitte "Clean Up" aus dem "Special"-Menü.

"Clean Up" steht für Aufräumen: Das bezieht sich natürlich nur auf den Amiga selbst. Besser gesagt, auf das durch diese Funktion angesprochene Window: Die Icons darin werden geordnet. Diese Funktion ist vor allem dann hilfreich, wenn in einem Window viele Icons kreuz und quer liegen. Durch "Clean Up" ist im Nu wieder Ordnung hergestellt.

Eine Erklärung sind wir Ihnen jetzt noch schuldig: Wie werde ich den Müll los, den ich nicht mehr brauche? Das wollen wir jetzt gleich beantworten, aber vorher noch eine wichtige Warnung:

Achtung: Auf Ihrer Workbench-Diskette ist nichts so unwichtig, daß Sie es wegwerfen sollten. Probieren Sie also bitte die "Empty Trash"-Funktion jetzt nicht aus. Außerdem ist es empfehlenswert, immer nur mit einer Kopie der Original-Workbench zu arbeiten. Wie Sie zu so einer Kopie kommen, erfahren Sie in Kapitel 3.7 im Amiga-Anwender-Handbuch. Auch von den anderen Disketten (Kickstart, Extras) sollten Sie Sicherheitskopien anfertigen. Wir gehen ab jetzt in unserem Buch davon aus, daß Sie mit Sicherheitskopien arbeiten.

Das endgültige Wegwerfen von Programmen und Dateien geht ganz einfach: Wenn Sie alles in den Trashcan befördert haben, aktivieren Sie das Trashcan-Icon und wählen dann aus dem Pulldown-Menü "Disk" die Funktion "Empty Trash". Jetzt wird alles, was sich im Mülleimer befindet, endgültig gelöscht. Übrig bleibt nur ein frisch geleerter Mülleimer.

Damit sind wir auch schon am Ende unseres kleinen Workbench-Ausflugs angelangt. Was Sie mit der Workbench außerdem noch anstellen können, finden Sie im Amiga-Anwender-Handbuch oder auch in unseren Büchern "Amiga 500 für Einsteiger" oder "Das große Amiga-2000-Buch".

Mit dem, was Sie in diesem Kapitel gelernt haben, sind Sie nun gut auf unsere Expedition ins AmigaBASIC vorbereitet. Lesen Sie nun bitte ab Kapitel 1 weiter. Es ist Ihnen natürlich auch nicht verboten, zusätzlich unser Einleitungskapitel für Fortgeschrittene und Profis zu lesen. Keine Sorge, wenn Sie darin noch nicht alles verstehen - alles wirklich Wichtige zur Workbench haben Sie bereits gelernt.

0.2 Workbench-Schnelldurchgang für Fortgeschrittene

Wenn Ihre Entscheidung auf diese Einleitung gefallen ist, sollten Sie schon ein wenig Erfahrung mit Computern haben oder sich zumindest ein bißchen mit dem Amiga auskennen. Wir wollen

hier kurz erklären, was beim Starten des Amiga passiert, und wir wollen auch die Grundfunktionen der Workbench erläutern. Wenn Sie das eine oder andere nicht verstehen, scheuen Sie sich nicht, auch das Kapitel 0.1 zu lesen.

Nach dem Einschalten führt der Amiga zunächst einen Selbsttest durch. Dazu gehört auch eine Überprüfung der Tonerzeugung: Sie hören eine kurze Tonfolge. Falls Sie nichts hören, besteht möglicherweise keine Verbindung der Tonausgänge zum Monitor-Lautsprecher bzw. einer Stereoanlage.

Nach diesem Test benötigt der Amiga 1000 die Kickstart-Diskette. Auf der Kickstart befindet sich das Betriebssystem des Amiga, das sogenannte *Kernel.* Das Kernel besteht aus einer großen Menge verschiedener Routinen, die von der Amiga-Software benutzt werden können (z.B. zur Bildschirmverwaltung, Steuerung von Ein- und Ausgaben und Überwachung des Multitasking). Der Inhalt der Kickstart-Diskette wird beim Amiga 1000 in einen 256 KByte großen RAM-Bereich geschrieben, der nach dem Laden elektronisch verriegelt wird und sich ab dann wie ROM verhält. Der Amiga 500 und der Amiga 2000 werden mit dem Kernel auf ROM-Bausteinen ausgeliefert, dadurch fällt das Laden der Kickstart-Diskette weg. Allerdings hat die Lösung, die beim Amiga 1000 verwendet wurde, den Vorteil, daß Verbesserungen des Betriebssystems einfach durch Austausch der Kickstart-Diskette möglich sind. Das wird bei Amiga 500 und 2000 nicht mehr so problemlos möglich sein.

Nach dem Laden der Kickstart (Amiga 1000) bzw. nach dem Einschalten (Amiga 500 & 2000) fordert der Amiga die Workbench-Diskette an. Von dieser Diskette wird die Benutzeroberfläche "Workbench" mit ihren verschiedenen Programmen geladen. Außerdem befinden sich auf dieser Diskette die AmigaDOS-Befehle, Systemprogramme (z.B. zur Druckeransteuerung oder Spracherzeugung) und verschiedene Utilities. Nach dem Laden kommen Sie direkt auf die Workbench, wo Sie den Amiga durch die Maus- und Windowtechnik sehr einfach bedienen können.

Der rote Mauszeiger bewegt sich synchron zur Bewegung der Maus auf dem Tisch. Die linke Maustaste dient allgemein zum Aktivieren und Starten, die rechte Taste wird zur Auswahl der Pulldown-Menüs verwendet.

Das Diskettensymbol in der rechten oberen Ecke des Bildschirms steht für die Workbench-Diskette. Ein solches Symbol heißt Icon. Wenn Sie den Mauszeiger auf das Workbench-Icon bewegen und mit der linken Maustaste zweimal klicken, wird die zugehörige Diskette geöffnet: Das Workbench-Window erscheint. Es enthält weitere Icons, die den Inhalt der Workbench-Diskette vertreten. Wenn Sie z.B. das Clock-Icon anklicken, wird ein Programm gestartet, das ein Window mit einer Uhr erzeugt.

Die Schubladen (Demos, Utilities, System, Empty) stellen Unter-Inhaltsverzeichnisse dar. Um deren Inhalt zu sehen, muß die jeweilige Schublade angeklickt werden. In den Schubladen können dann wieder Schubladen sein usw.

Am Rahmen der Windows sehen Sie verschiedene Symbole. Wenn Sie sich mit diesen Symbolen nicht auskennen, schauen Sie sich bitte Bild 1 und den dazugehörenden Text im Kapitel 0.1 an. Die Symbole sind sehr wichtig für die Arbeit mit Windows und werden dort ausführlich erklärt.

Mit der rechten Maustaste bedienen Sie die Pulldown-Menüs. Deshalb heißt diese Taste "Menütaste". Solange Sie die Menütaste gedrückt halten, stehen in der Kopfleiste die Titel der verfügbaren Pulldowns. Wenn Sie den Mauszeiger mit gedrückter rechter Taste auf einen dieser Titel bewegen, erscheint das zugehörige Pulldown. Um einen Menüpunkt auszuwählen, bewegen Sie den Zeiger auf die gewünsche Funktion und lassen dann los. Nur die gut lesbaren Punkte stehen zur Auswahl, die anderen, schlechter lesbaren, sind zur Zeit nicht aktiv. Dasselbe gilt übrigens auch für Windows: Aktive Windows erkennen Sie an ihrem deutlich lesbaren Titel.

Die Funktionen "Open" und "Close" im "Workbench"-Pulldown bieten eine weitere Möglichkeit, Icons bzw. Windows zu öffnen und zu schließen. "Duplicate" dient zum Kopieren von Pro-

grammen und Disketten, "Rename" zum Umbenennen und "Discard" zum endgültigen Löschen. "Info" öffnet ein Informations-Window, in dem Sie Daten wie Größe des Programms, Programmtyp, Löschschutz und Bemerkungen zum Programm ansehen können. "Empty Trash" aus dem "Disk"-Menü löscht den Trashcan, dazu gleich mehr. "Initialize" formatiert Disketten.

Zum "Special"-Menü: "Clean Up" räumt ungeordnete Windows auf, "Last Error" wiederholt die letzte Meldung des Betriebssystems. "Redraw" baut den Bildschirm neu auf, falls durch einen Programmfehler Teile gelöscht wurden. "Snapshot" fixiert aktivierte Icons an ihrer Bildschirmpositon und "Version" gibt die Versionsnummer der Workbench aus.

Nach diesem Schnelldurchgang durch die Pulldown-Optionen noch ein Thema: Der Trashcan. In den Trashcan (deutsch: Mülleimer) werfen Sie Programme, die gelöscht werden sollen. Der Trashcan ist im Grunde nichts weiter als ein besonderes Inhaltsverzeichnis. Erst wenn Sie ihn aktiviert haben und die Pulldown-Funktion "Empty Trash" anwählen, wird sein Inhalt gelöscht. Dieses Löschen ist dann allerdings unwiderruflich. Schon aus diesem Grund sollten Sie immer nur mit Sicherheitskopien Ihrer Original-Disketten arbeiten. Wie Sie solche Sicherheitskopien herstellen können, wird im Amiga-Anwender-Handbuch im Kapitel 3.7 beschrieben. Wir gehen ab jetzt in diesem Buch davon aus, daß Sie mit Sicherheitskopien Ihrer Disketten arbeiten.

Und damit wären alle Vorbereitungen für die Arbeit mit AmigaBASIC getroffen. Wir können anfangen. Das heißt, etwas fehlt noch, bevor wir richtig einsteigen können ins AmigaBASIC.

Zwischenspiel 1: Formalitäten - Vorbereitung für die Diskette im Buch

Zwischenspiele wie dieses hier werden Sie im Buch immer wieder antreffen. Zwischenspiele nehmen wichtige Informationen auf, die aber eigentlich nicht direkt in den Ablauf passen.

Bevor wir mit Ihnen gemeinsam die ersten Versuche mit AmigaBASIC machen, müssen Sie noch eine Vorbereitung treffen. Es dreht sich dabei um unsere Diskette im Buch. Was es damit auf sich hat, und wann Sie sie verwenden können, haben wir ja schon in unserem Einleitungskapitel beschrieben.

Bevor Sie die Programme darauf jedoch durch direktes Anklicken starten können, müssen Sie erst noch das "AmigaBASIC" von der "ExtrasD"-Diskette, die Ihnen Commodore geliefert hat, auf unsere Diskette kopieren. Aus rechtlichen Gründen war es uns nämlich leider nicht möglich, die Diskette im Buch komplett mit BASIC auszuliefern.

Aber keine Sorge. Was Sie dabei tun müssen, ist ganz einfach:

- Bitte legen Sie die Sicherheitskopie Ihrer "ExtrasD"-Diskette in ein Laufwerk und öffnen Sie sie.

Klicken Sie also zweimal auf das Disketten-Icon. Daraufhin öffnet sich ein Window, in dem verschiedene weitere Icons erscheinen. Vermutlich ganz links oben in diesem Window werden Sie das Programm "AmigaBASIC" entdecken. Sein Icon ist ein weißer Quader mit orangen Symbolen auf dem Vordergrund.

Noch schnell ein Satz zur Sicherheitskopie: Am Ende des letzten Kapitels haben wir Sie gebeten, von Ihrer "ExtrasD"-Diskette eine Sicherheitskopie anzufertigen. Das haben Sie ja auch sicher prompt erledigt. Oder sollte tatsächlich noch jemand dabei sein, der... Na, dann wird's aber Zeit. Falls das noch nicht geschehen ist, fertigen Sie bitte gleich jetzt eine Sicherheitskopie Ihrer Original-"ExtrasD" an. Wie das geht, finden Sie im Amiga-Benutzerhandbuch im Kapitel 3.7.

Und weil wir gerade dabei sind: Auch unsere Diskette im Buch ist genau genommen eine Originaldiskette. Deshalb sollten Sie bitte auch von ihr eine Sicherheitskopie anfertigen und nur mit dieser Kopie arbeiten. Wenn Sie nämlich auf Ihrem einzigen Exemplar dieser Diskette versehentlich etwas löschen oder ein

Problem mit einem Diskettenlaufwerk auftritt, dann haben Sie auch hier Arbeit und Zeitverlust, um wieder ein Original zu besorgen.

So, jetzt aber weiter im Text.

- Wenn Sie zwei Diskettenlaufwerke besitzen, legen Sie bitte ins andere Laufwerk unsere Diskette im Buch und öffnen Sie sie ebenfalls.

Unsere Diskette hat übrigens den Namen "BasicDisk".

- Wenn Sie nur ein Diskettenlaufwerk haben, ist das auch kein Problem. Bitte lassen Sie in diesem Fall die "ExtrasD"-Diskette im Laufwerk und öffnen Sie die RAM-Disk.

- Wenn Sie zwei Laufwerke besitzen, verschieben Sie mit der Maus nun bitte das Icon von "AmigaBASIC" auf der "ExtrasD"-Diskette in das Window unserer "BasicDisk".

Wir haben in der linken oberen Ecke des Windows einen Platz für AmigaBASIC freigehalten.

- Bei nur einem Laufwerk schieben Sie das AmigaBASIC-Icon bitte ins Window der RAM-Disk.

Bitte warten Sie nun, bis die roten Laufwerkslampen bei allen Laufwerken erloschen sind.

Bei zwei Laufwerken sind Sie nun fertig.

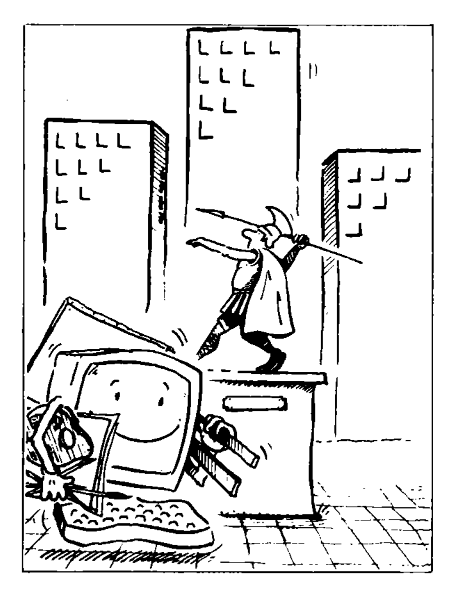
- Wenn Sie nur ein Laufwerk besitzen, entnehmen Sie bitte die "ExtrasD"-Diskette und legen Sie dann unsere "BasicDisk" ein.

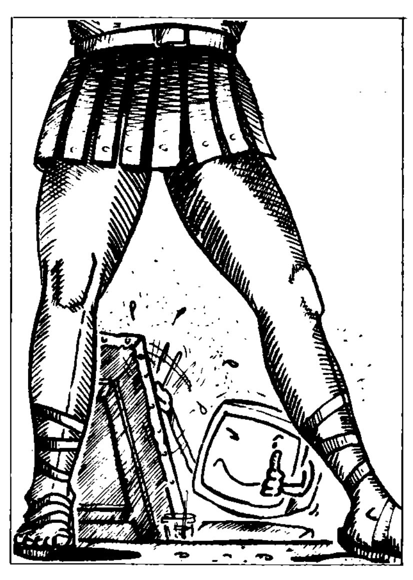
- Öffnen Sie dann das Window unserer Diskette, schieben Sie das AmigaBASIC-Icon von der RAM-Disk in das Window der "BasicDisk" und warten Sie wieder, bis die Laufwerkslampe erloschen ist.

- Wenn Sie über die RAM-Disk kopiert haben, aktivieren Sie bitte das AmigaBASIC-Icon dort (einmal klicken) und wählen Sie aus dem "Workbench"-Window die Option "Discard". Damit wird der Platz auf der RAM-Disk wieder freigegeben.

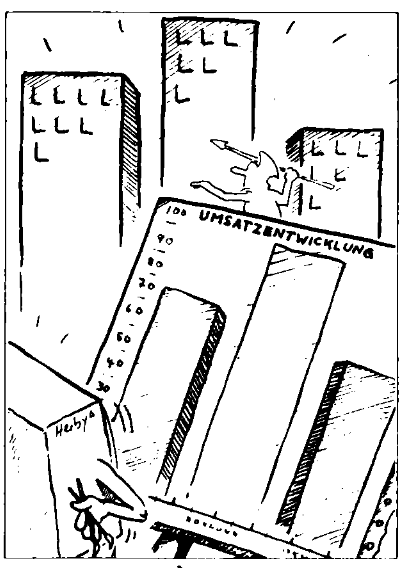
Soweit unsere Startvorbereitungen. Sie können unsere Diskette nun wieder aus dem Laufwerk entnehmen. Im nächsten Kapitel benötigen Sie zunächst die Sicherheitskopie Ihrer "ExtrasD"-Diskette. Und damit geht's dann endgültig los.

I. Der Grafik-Amiga









1. Es bewegt sich was - Objekt-Animation

In den folgenden Kapiteln dreht sich alles um die Grafikmöglichkeiten des Amiga. Gerade auf diesem Gebiet haben Sie sicher schon einiges gesehen. Vieles davon ist auch in BASIC zu realisieren - mit sehr einfachen Befehlen.

Im ersten Teil dieses Buches zeigen wir Ihnen, was AmigaBASIC zum Thema Animation, also zum Thema bewegte Grafik, zu bieten hat. Wenn Sie unsere Beispiele mit abtippen, haben Sie am Ende sogar ein kleines Programm, mit dem Sie bewegte Titelbilder für Videos erzeugen können. Aber auch, wenn Sie keinen Videorecorder besitzen, werden Ihnen die Ergebnisse gefallen. Schnallen Sie sich an - im folgenden geht es um fliegende Bälle und wandernde Sterne - oder eben um: Computeranimation in BASIC.

1.1 Amiga lernt - die Extras-Diskette

Zuerst einmal müssen wir dem Amiga aber beibringen, was BASIC überhaupt ist. Er soll ja zum Schluß etwas anfangen können mit Befehlen wie GOTO oder IF...THEN. Moment mal, werden Sie vielleicht jetzt sagen: Wie kann ich einem Computer etwas beibringen, was ich selber noch nicht kann? Ganz einfach: Während Sie dieses Buch lesen müssen, um BASIC zu verstehen, hat es der Amiga da leichter. Er braucht nur eine Diskette namens "ExtrasD" - in der findet er alles, was für ihn wichtig ist.

Suchen Sie bitte aus den mitgelieferten Disketten diejenige mit der Aufschrift "Amiga Extras D 1.2". Wenn Sie glücklicher Besitzer eines zweiten Laufwerks sind, legen Sie die Extras-Diskette bitte in das externe Laufwerk. Andernfalls entnehmen Sie die Workbench und schieben Sie "Extras" in das interne Laufwerk.

Achtung: Da besonders Besitzer eines einzigen Laufwerks bereits jetzt Diskjockey-Tätigkeiten ausüben müssen, eine Warnung an alle: Disketten dürfen erst dann aus dem Laufwerk ge-

> nommen werden, wenn die rote Lampe am Disketten-Laufwerk erloschen ist. Sie laufen sonst Gefahr, den Inhalt der gesamten Diskette zu zerstören! Auch denjenigen, die sich schon für Computer-Profis halten, sei das nachdrücklich gesagt: Bei manchen Computern kann man während des Lesens die Disketten entnehmen. Der Amiga reagiert darauf ausgesprochen empfindlich! Rote Lampe heißt nun mal "Diskette nicht entnehmen". Und Ihr Amiga meint, was er sagt!

Auf dem Workbench-Bildschirm sehen Sie nun die beiden Symbole für die Disketten "Workbench 1.2" und "ExtrasD". (Das gilt auch dann, wenn Sie die Workbench-Diskette aus dem Laufwerk genommen haben. Zu dieser Diskette hat der Amiga eine besondere innige Beziehung. Deshalb vergißt er sie nie!)

- Öffnen Sie "Extras".

Das heißt, bewegen Sie den Mauspfeil auf das Symbol der Extras-Diskette und betätigen Sie zweimal kurz die linke Maustaste. Auf dem Bildschirm erscheint ein Window, das etwa sieben Icons enthält: Hinweise, Tools, FD1.2, Terminal, Trashcan, AmigaBASIC und BasicDemos. Uns interessieren jetzt vor allem die letzten beiden.

Zu den anderen Icons sei an dieser Stelle nur soviel gesagt: "Hinweise" ist ein Text, den Sie sich mit dem Notepad der Workbench-Diskette ansehen können. Sie finden dort einige Bemerkungen zur vorliegenden Version der Extras-Diskette. Die "Tools"-Schublade beinhaltet einige Programme, die wir Ihnen im Anhang C genauer vorstellen werden. Dasselbe gilt für das Programm "Terminal". Zum Inhalt der Schublade "FD1.2" kommen wir später, Sie dürfen diese Schublade auf keinen Fall löschen!

Der weiße Quader mit den orangen Symbolen auf der Vorderseite ist das Icon zu AmigaBASIC. Klicken Sie ihn an. Nach kurzer Zeit ist AmigaBASIC geladen und Sie sehen folgendes Bild:

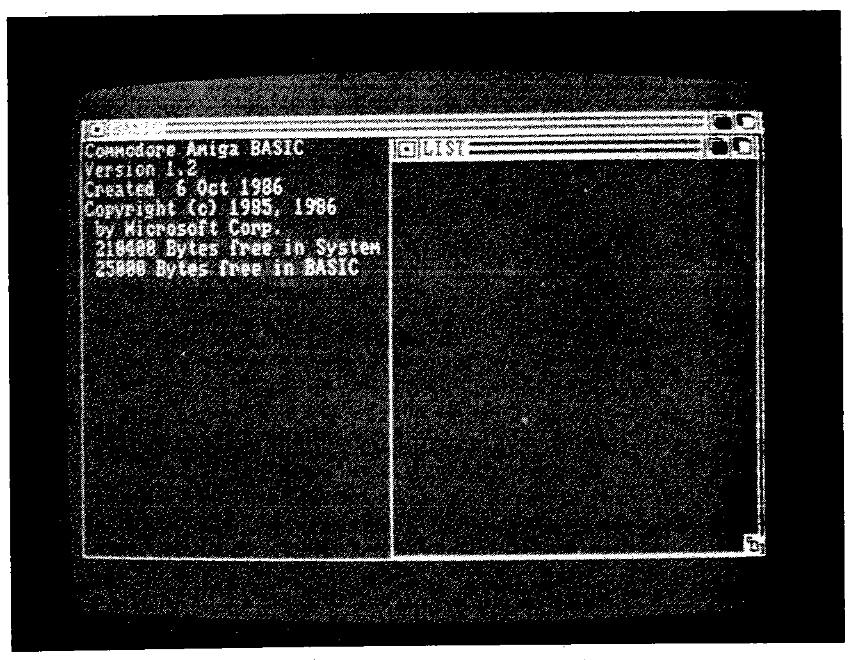


Bild 2: So meldet sich AmigaBASIC

1.2 Dürfen wir vorstellen - AmigaBASIC

Damit hätten wir AmigaBASIC erstmal geladen. Aber was soll dieser Bildschirm denn nun bedeuten? Wir sehen da zwei Windows: Eines heißt "BASIC", das andere heißt "LIST". Das LIST-Window ist aktiv (das erkennt man an der klar lesbaren Schrift - erinnern Sie sich?), aber ziemlich leer. Im anderen Window, mit dem Titel BASIC, steht zu lesen:

```
Commodore Amiga BASIC

Version 1.2

Created  6 Oct 1986

Copyright (c) 1985, 1986

 by Microsoft Corp.

 261048 Bytes free in System

 25000 Bytes free in BASIC
```

Falls sich die eine oder andere Angabe von dem, was Sie bei sich auf dem Bildschirm lesen, unterscheidet, macht das auch nichts. In "Commodore Amiga BASIC" werden wir uns wohl nicht unterscheiden. So heißt unser BASIC nämlich.

Bei der Versionsnummer kann bei Ihnen schon etwas völlig anderes stehen. Softwarefirmen versehen ihre Produkte immer mit Versionsnummern. Daran kann man dann leicht erkennen, auf welchem Stand das jeweilige Programm ist. Denn mit Programmen ist das so eine Sache: Selbst wenn die Programmierer das Gefühl haben, nun sei alles fertig - kaum ist das Werk veröffentlicht, finden sich Fehler, die vorher übersehen wurden. AmigaBASIC macht da keine Ausnahme. Wir werden deshalb in diesem Buch auch an verschiedenen Stellen auf Fehler, die es bei AmigaBASIC noch gibt, hinweisen.

Die erste veröffentlichte Version heißt meistens 1.00. Manchmal sieht man auch Vorab-Veröffentlichungen von Programmen mit Versionsnummern wie 0.99. Das heißt dann, daß die Programme zwar noch nicht ganz fertig sind, aber ganz kurz vor ihrer Vollendung stehen. Wenn kleinere Fehler behoben wurden, folgen bald die Versionen 1.01, 1.02 etc. Wurde dann etwas mehr geändert, ist Version 1.1 dran. Um eine Version 2.0 zu rechtfertigen, müssen schon große Änderungen oder starke Erweiterungen durchgeführt worden sein. Aber dabei passieren natürlich

auch wieder Fehler, und deshalb läßt auch Version 2.01 nicht lange auf sich warten. Und so geht das weiter. Es gibt Programme, von denen es die Version 5.21 gibt. So hohe Versionsnummern heißen aber nicht unbedingt, daß das Programm am Anfang sehr fehlerhaft war. Sie sprechen vielmehr für großen Erfolg, denn sonst hätte man sich nicht so viel Mühe mit der Verbesserung und den Erweiterungen gemacht.

Ihr AmigaBASIC sollte die Versionsnummer 1.2 haben. Die Entwickler von Commodore haben die Version 1.1 mehr oder weniger unter den Tisch fallen lassen, um sich in der Bezeichnung den Workbench- und Kickstartversionen anzupassen. Mit AmigaBASIC 1.0 sollten Sie nicht mehr arbeiten; wenn Sie kein anderes AmigaBASIC haben, besorgen Sie sich die neueste Version der "ExtrasD"-Diskette.

In der nächsten Zeile der Einschaltmeldung lesen Sie "Created 6 Oct 1986". Die Programmierer von AmigaBASIC haben den letzten Tastendruck für diese Version am 06.10.1986 getan. In späteren Versionen kann dieses Datum geändert oder durch weitere Daten ergänzt werden.

"Copyright (c) 1985, 1986 by Microsoft Corp." Die amerikanische Softwarefirma Microsoft Corporation hat also unser BASIC geschrieben. Microsoft hat schon für viele Computer sehr gute und erfolgreiche BASICs hergestellt. Der IBM PC, der Apple Macintosh, fast alle Commodores (vom alten PET bis zum Commodore 128) - alle haben sie ihr BASIC von dieser Firma. Auch der Amiga macht da keine Ausnahme. Und weil das Schreiben eines BASIC-*Interpreters* (so sagt der Fachmann dazu; wenn's Ihnen nichts sagt - bitte im Fachwortlexikon im Anhang D nachlesen) viel Mühe macht, erhebt die Firma auch Urheberschutz (engl.: Copyright) darauf.

Kommen wir zu den beiden letzten Angaben. Und die können sich jetzt stark von unserer obigen Meldung unterscheiden. Denn hier geht es um Speicherplatz. Speicherplatz wird allgemein in *Bytes* angegeben. Was ein Byte genau bedeutet, können Sie ebenfalls im Fachwortlexikon finden.

Die erste Zahl gibt den freien Speicher im "System" an. Das heißt, die Anzahl an Bytes, die neben AmigaBASIC, dem Speicherbereich für BASIC-Daten und dem Speicherplatz, der für andere Aufgaben verwendet wird, momentan noch frei ist.

Je nachdem, was Sie vorher mit dem Amiga getan haben, wieviele Disketten und wieviele Windows geöffnet wurden, und ob vielleicht andere Programme im Hintergrund laufen, kann sich dieser Wert sehr stark von unseren 261048 Bytes unterscheiden. Außerdem sollten wir noch dazusagen, daß wir einen Amiga 1000 mit 256 KByte-Erweiterung benutzen. Wir haben also insgesamt 512 KByte *RAM* (d.h.: Schreib-/Lesespeicher). Beim Amiga 500 oder 2000 kann der freie Speicherplatz, je nach Ausbaustufe, wesentlich größer sein. Aber mit 512 KByte RAM kommen Sie für die Programme, die wir in diesem Buch vorstellen, in jedem Fall aus. Die zweite Zahl gibt die Anzahl an Bytes an, die BASIC für seine Programme und Daten zur Verfügung hat. Das sind im Normalfall 25000 Bytes.

Jetzt werden Sie sich vielleicht fragen, warum AmigaBASIC relativ wenig von dem "System"-Speicher für sich nutzt, der doch offenbar im Überfluß vorhanden ist? Dafür gibt es mehrere Antworten: Durch das *Multitasking* können andere Programme auch noch Speicherplatz beanspruchen. Außerdem brauchen die farbigen Grafiken, die der Amiga darstellen kann, viel Speicherplatz. Damit Sie im Bedarfsfall auch mehrere hochauflösende Grafiken in der höchstmöglichen Anzahl an Farben im Speicher unterkriegen (und die brauchen immerhin pro Grafik 128 KBytes), ist der freie Systemspeicher im Vergleich zum BASIC-Speicher sehr großzügig bemessen. Im BASIC-Referenzteil werden wir Ihnen beim CLEAR-Befehl zeigen, wie Sie die Speicheraufteilung nach Ihren Vorstellungen verändern können.

1.3 Hallo Amiga - Experimente mit AmigaBASIC

Das BASIC-Window haben wir uns jetzt sehr genau angesehen. Sonst gibt es da erstmal nichts weiter zu entdecken. Aber schauen wir uns noch mal genauer das LIST-Window an. So ganz leer, wie ursprünglich behauptet, ist es ja gar nicht. In der

linken oberen Ecke des Windows sehen Sie einen orangefarbigen Strich. Dieser Strich ist der BASIC-Cursor, der sich zur Zeit im LIST-Window befindet.

Trotzdem - das BASIC-Window ist uns mittlerweile etwas vertrauter. Also wollen wir mal sehen, ob man damit nicht ein bißchen was anfangen kann. Es ist bisher nicht aktiv, aber dem kann ja abholfen werden:

- Bringen Sie den Mauscursor an irgendeine Stelle des BASIC-Windows, und klicken Sie einmal mit der linken Taste.

Sieh an! Unser Computer hat uns verstanden. Zumindest behauptet er das mit seinem OK, das er auf den Bildschirm geschrieben hat. Und der BASIC-Cursor steht jetzt auch im anderen Window. Ein kurzer Blick auf den Window-Titel beweist: das BASIC-Window ist aktiv. Wenn wir jetzt ins LIST-Window klicken, haben wir dieses Window wieder aktiviert. Nochmal ins BASIC-Window - aha, er hat's wieder verstanden.

Bisher verhält sich unser Amiga so, wie wir es auch von der Workbench gewohnt sind. Es gibt hier zwei Windows, von denen immer nur eines aktiviert sein kann.

Der erste Unterschied besteht in der Anzahl der Cursors, die wir zur Verfügung haben. Davon gibt es jetzt nämlich zwei. Der Maus-Cursor ist, wie der Name schon sagt, für die Maus zuständig. Und der andere? Er heißt, wie schon erwähnt, BASIC-Cursor und wird bei Tastatureingaben benötigt. Er zeigt, an welcher Stelle des Bildschirms die eingetippten Zeichen erscheinen. Diesen Cursor benötigen wir in BASIC häufig, weil beim Programmieren viel mit der Tastatur gearbeitet wird. Sollten Sie die Tastatur unter Ihrem Amiga versteckt haben, wird es also jetzt Zeit, sie aus der Versenkung zu holen. Damit Sie bequem tippen können, empfehlen wir Ihnen, die Füßchen aufzuklappen, so daß die Tastatur mit leichter Neigung vor Ihnen steht.

Nach all diesen Vorbereitungen sind wir soweit für unseren ersten Versuche mit AmigaBASIC. Tippen Sie:

```
print "Hallo"
```

Die obere Zeile sollte jetzt genauso auf dem Bildschirm stehen. Falls Sie sich vertippt haben, können Sie die Zeichen links vom Cursor mit der Taste <BACKSPACE> wieder löschen. Wenn Sie soweit sind, drücken Sie bitte die Taste <RETURN>. Diese Taste wird auch Eingabetaste genannt. Was für einen Sinn hat diese Taste? Der Amiga kann nicht wissen, wann Sie mit dem Tippen auf dem Bildschirm fertig sind. Deshalb kümmert er sich erstmal gar nicht um das, was auf dem Bildschirm steht. Erst durch das Drücken der <RETURN>-Taste sagen Sie ihm: "So, lieber Amiga, nun schau Dir an, was ich da geschrieben habe."

Das tut der Amiga auch und bringt Ihnen als Antwort ein freundliches "Hallo" entgegen, gefolgt von seinem bekannten OK. Mit unserer Eingabe haben wir ihm aufgetragen, daß der Text, der hinter dem PRINT-Befehl in Anführungszeichen steht, auf dem Bildschirm gedruckt wird (engl.: print). Dabei ist es dem Amiga egal, ob Sie print oder PRINT schreiben. Groß- und Kleinschreibung ist bei BASIC-Befehlen unwichtig. Wenn es Ihnen Spaß macht, können Sie sogar pRiNt schreiben. Als Zeichen dafür, daß der Amiga mit allen Aufgaben fertig ist, folgt noch ein OK.

Jetzt ist es auch nicht mehr schwierig zu beantworten, was wohl der folgende Befehl bewirken wird:

```
print "Wie geht's?" <RETURN>
```

Vorsicht: Sie sollen das Obenstehende jetzt nicht exakt abtippen, sondern diese Schreibweise bedeutet, daß hinter dem Befehl print "Wie geht's?" noch die <RETURN>-Taste gedrückt werden muß.

Höfliche Fragen sollte man höflich beantworten. Also geben Sie ein:

```
Danke, mir geht es gut. <RETURN>
```

Sie werden gleich merken, daß man mit einem Computer nicht so einfach Konversation machen kann. Zunächst beschwert er sich einmal lautstark mit einem Pieps, dann erscheint ein Kästchen mit dem Text "Undefined Subprogram" und einem Feld "OK".

Vielleicht haben Sie es sich schon gedacht: Das Ganze ist eine Fehlermeldung. Der Amiga gibt uns zu verstehen, daß der eingegebene Satz ihm nicht viel sagt. Ist ja auch kein Wunder, er versteht ja nur BASIC und "Danke, mir geht es gut." ist zwar nett, aber mit Sicherheit kein BASIC-Befehl. Um dem Amiga mitzuteilen, daß wir seine Fehlermeldung verstanden haben, müssen wir mit dem Mauscursor in das OK-Feld klicken. Daraufhin verschwindet das Fehler-Kästchen. Am besten, Sie lassen den Mauscursor gleich in dieser Gegend stehen. Denn erfahrungsgemäß produziert man beim ersten Experimentieren in BASIC ziemlich viele Errors.

Die nächste Frage, die sich stellt, ist, warum man eigentlich die Anführungszeichen braucht? Wir können's ja mal ohne versuchen:

```
print Hallo <RETURN>
```

Ihr Amiga wird Ihnen antworten:

```
0
OK
```

Wie ist diese 0 jetzt zu verstehen? Will der Amiga uns in im Sinne von "Null, kannste vergessen!" mitteilen, daß er überhaupt keine Lust hat, unserer Anforderung nachzukommen, wenn wir uns nicht an die Regel mit den Anführungszeichen halten? Oder ist die Null etwa als Beleidigung gemeint? Bevor Sie Ihren Amiga verärgert einpacken und wegen unverschämten Benehmens zum Händler zurückbringen, sollten Sie die folgenden Zeilen lesen. Tippen Sie ein:

```
let Hallo = 100 <RETURN>
print Hallo <RETURN>
```

Diesmal wird uns der Amiga keine 0, sondern die Zahl 100 anbieten. Hat schon jemand einen Verdacht? Vorhin haben wir geschrieben, daß Texte, die dem PRINT-Befehl folgen, in Anführungszeichen eingebettet werden müssen. Demnach ist das Hallo im oberen Fall kein Text. Es handelt sich dabei vielmehr um eine Variable. Ja richtig, um eines dieser Dinger, die im Mathematikunterricht schon immer so viele Probleme machten, weil sie meistens unbekannt sind. In BASIC ist das nicht so schwierig.

Eine der Aufgaben eines BASIC-Programms kann es sein, Berechnungen durchzuführen. Um sich die Werte und Ergebnisse solcher Berechnungen zu merken, braucht BASIC aber einen Namen für das Kind. Dazu dienen die Variablen. Und der LET-Befehl weist den Variablen einen Wert zu. Geben Sie bitte ein:

```
print 7+5 <RETURN>
```

Der Amiga wird Ihnen als Ergebnis die Zahl 12 liefern. Wenn Sie jetzt

```
let Hallo = 7+5
print Hallo
```

eingeben, wird der Amiga Ihnen dasselbe Ergbnis liefern. Aber diesmal hat er sich das Ergebnis der Berechnung in einer Variablen namens Hallo gemerkt. Wenn Sie das nicht glauben, können Sie's ja nochmal nachprüfen:

```
print Hallo <RETURN>
```

Wieder wird das Ergebnis 12 sein. Und so kam auch die 0 vorhin zustande: Wir haben den Inhalt einer Variablen abgefragt, die es überhaupt noch nicht gab. Logischerweise kam als Ergebnis 0 heraus, denn alle Variablen haben erstmal als Grundbelegung den Wert 0.

Übrigens, wenn Ihnen die Tipperei allmählich etwas zu viel wird, hier sind zwei Tips, die Ihnen auf Dauer eine ganze

Menge Arbeit abnehmen. Erstens: Den Befehl LET können Sie auch weglassen:

```
let Hallo = 5
```

und

```
Hallo = 5
```

bewirken genau dasselbe. Zweitens: Für den PRINT-Befehl, der ja offensichtlich sehr häufig benötigt wird, gibt's eine Abkürzung, nämlich das Fragezeichen (?).

```
print "Hallo"
```

und

```
? "Hallo"
```

bewirken auch dasselbe, nur daß man sich mit der Abkürzung einiges an Tipperei spart.

Alles, was wir bisher gelernt haben, läßt sich natürlich auch kombinieren, etwa in der Art:

```
? "Das Ergebnis von 5+7 ist" 5+7
```

Allerdings werden Sie beim Eingeben wahrscheinlich ein Problem haben: Etwa nach dem zweiten Anführungszeichen ist Ihnen das LIST-Window im Weg. Und Ihr BASIC-Cursor verschwindet auf einmal dahinter, ohne Sie auch nur eines Abschiedswortes zu würdigen. Mit unseren Workbench-Kenntnissen ist es uns aber ein leichtes, das BASIC-Window in den Vordergrund zu holen: Einfach das rechte der beiden Vordergrund-/Hintergrund-Symbole am BASIC-Window anklicken. (Zur Hilfestellung: Das ist das Symbol ganz rechts oben in der Ecke.) Schon füllt das BASIC-Window den ganzen Bildschirm aus und wir haben genug Platz für unsere Eingaben. Also weiter im Text:

```
? "Das Ergebnis von 5+7 ist" 5+7 <RETURN>
```

Na, das sieht doch ganz gut aus, oder? Wahrscheinlich ist Ihnen schon aufgefallen, daß das BASIC-Window durch unsere Experimente etwas unaufgeräumt aussieht. Überall sind Fragmente unserer vorherigen Versuche. Auch dagegen gibt es Abhilfe:

```
cls <RETURN>
```

Sieht gleich viel ordentlicher aus, oder? Der CLS-Befehl löscht den Bildschirm (engl.: Clear Screen) und setzt den Cursor ins linke obere Eck. Das ist immer dann sinnvoll, wenn der Bildschirm schön sauber aussehen soll, damit es genug Platz gibt für das, was danach passieren soll. Wenn Sie sicherstellen wollen, daß der Bildschirm gelöscht wird, bevor eine Ausgabe erfolgt, erreichen Sie das so:

```
cls : ? "Hallo!" <RETURN>
```

Damit haben wir gleich wieder etwas gelernt: Man kann mehrere BASIC-Befehle in eine Zeile schreiben, indem man sie mit einem Doppelpunkt voneinander trennt.

Wenn Sie sich jetzt sagen, "Das ist ja alles gut und schön, aber ein bißchen mehr Aktion wäre schon nicht schlecht!", dann kann Ihnen geholfen werden: Als Kostprobe, was bei AmigaBASIC mit Animation zu erreichen ist, wollen wir Ihnen zunächst ein kleines Programm vorstellen, das ein paar fliegende Bälle erzeugt. Der erste Schritt dazu wäre, den Namen des Programms auf den Bildschirm schreiben zu lassen.

```
? "Fliegende Bälle" <RETURN>
```

Jetzt steht zwar der Titel unseres zukünftigen Werks auf dem Bildschirm, aber so ganz glücklich sind Sie wahrscheinlich noch nicht darüber: Das bisher Getippte hat sich ja als ziemlich vergänglich erwiesen - spätestens, wenn es am oberen Bildschirmrand verschwunden ist, weil Sie neue Eingaben gemacht haben.

Aber es kann ja wohl kaum Sinn von BASIC sein, ein Programm nach jedem Lauf neu abzutippen, oder? Richtig. Deshalb kommt jetzt die große Stunde des LIST-Windows.

1.4 Die erste Kostprobe: Nur Fliegen ist schöner

Das LIST-Window liegt ja zur Zeit etwas unbeachtet hinter dem BASIC-Window, mit dem wir die ganze Zeit gearbeitet haben. Ein Klick ins Hintergrund-Symbol (das linke der beiden Symbole) befördert es wieder zu Tage. Genauer gesagt, wird das BASIC-Window ganz nach hinten gelegt. Deshalb werden Sie jetzt wahrscheinlich zwei neue Windows sehen: Das gesuchte LIST-Window und das Window der Extras-Diskette. Letzteres können wir zur Zeit wirklich nicht brauchen. Es muß also wieder verschwinden. Am besten, Sie schließen es ganz (ein Klick ins Schließ-Symbol, links oben am Window).

So, wenn jetzt alle soweit sind, können wir anfangen, unser erstes eigenes Programm zu schreiben. Es ist wirklich ganz einfach. Aktivieren Sie bitte das LIST-Window (irgendwo ins Window klicken). Jetzt verschwindet der BASIC-Cursor aus dem BASIC-Window und erscheint im LIST-Window. Tippen Sie:

```
? "Fliegende Bälle" <RETURN>
```

Der Amiga reagiert nun anders als bisher gewohnt: Anstatt den Befehl direkt auszuführen, hat er aus dem ? ein PRINT gemacht. Der Cursor steht eine Zeile tiefer und wartet auf weitere Eingaben. Na, damit können wir dienen:

```
? "von Franz Maier" <RETURN>
```

Natürlich können Sie hier Ihren eigenen Namen einsetzen. Haben Sie keine Skrupel, sich ein wenig mit fremden Federn zu schmücken: Schließlich soll Ihnen für die ganze Arbeit, die Sie bisher hatten, ja auch ein wenig Ruhm zuteil werden. (Ihre Familie sollten Sie aber erst rufen, wenn das Programm abgetippt ist und läuft!!) Der Amiga reagiert wieder genauso wie gerade

eben auf unsere Eingabe: Er übersetzt das ? und kümmert sich nicht weiter um das Getippte.

Es gibt noch weitere Punkte, in denen sich das LIST-Window vom BASIC-Window unterscheidet. Drücken Sie zum Beispiel mal eine der vier Cursor-Tasten. (Das sind die vier Tasten mit den Pfeilen darauf, die rechts unter bzw. neben der <RETURN>-Taste auf der Tastatur angeordnet sind.) Je nachdem, welche Taste Sie erwischt haben, bewegt sich der Cursor im LIST-Window. Wenn er sich nicht bewegt, und der Amiga stattdessen einen protestierenden Piepser von sich gibt, haben Sie versucht, den Cursor in eine Richtung zu bewegen, in der es für ihn zur Zeit kein Fortkommen gibt. Die Pfeile auf den Tasten entsprechen der Richtung, in die sich der Cursor bewegt.

Mit den Cursor-Tasten und der <BACKSPACE>-Taste können Sie Tippfehler an jeder Stelle innerhalb des Windows korrigieren. Wenn die <BACKSPACE>-Taste auf Ihrer Tastatur nicht beschriftet ist, dann erkennen Sie sie daran, daß sie direkt über der <RETURN>-Taste liegt und einen Pfeil nach links trägt. Die Korrekturmöglichkeiten verdanken Sie dem *Bildschirm-Editor*. Der heißt so, weil Sie den Cursor frei auf dem ganzen Bildschirm bewegen können. Im BASIC-Window haben Sie im Gegensatz dazu einen *Zeilen-Editor*, weil Korrekturen nur innerhalb der aktuellen Zeile erfolgen können.

Geben Sie nun bitte folgende Programmzeilen im LIST-Window ein:

```
p$ = "ExtrasD:BasicDemos/Ball" <RETURN>
open p$ for input as 1 <RETURN>
object.shape 1,input$(lof(1),1) <RETURN>
close 1 <RETURN>

Start: <RETURN>
for x=2 to 5 <RETURN>
object.shape x,1 <RETURN>
object.x x,320 <RETURN>
object.y x,60 <RETURN>
object.hit x,0,0 <RETURN>
```

```
object.ax x, ((x=2 or x=4)+.5)*6 <RETURN>
object.ay x, ((x>3)+.5)*1.5 <RETURN>
next x <RETURN>

for x=2 to 5 <RETURN>
object.on x : object.start x <RETURN>
next x <RETURN>

for x=1 to 3500 : next x <RETURN>

goto Start <RETURN>
```

Bravo! Sie haben soeben Ihr erstes AmigaBASIC-Programm geschrieben. Bitte achten Sie darauf, daß Sie alles so eingeben, wie es oben abgedruckt ist. Der Amiga wird die BASIC-Befehle beim Drücken der <RETURN>-Taste zwar in Großbuchstaben umwandeln, aber ansonsten sollte alles genau übereinstimmen. Wenn Sie Fehler finden, wissen Sie ja: Mit den Cursor-Tasten und <BACKSPACE> kann man sie korrigieren.

Klicken Sie jetzt ins BASIC-Window und geben Sie ein:

```
run <RETURN>
```

Wenn Sie sich nicht vertippt haben, sehen Sie jetzt vier kleine Bälle, die von der Mitte des Bildschirms in alle Richtungen davonfliegen. Es kann aber auch sein, daß der Amiga einen protestierenden Piepser von sich gibt und das schon bekannte Kästchen im oberen Teil des Bildschirms erscheint. In diesem Fall haben Sie doch noch einen Fehler in Ihrer Eingabe übersehen. Bewegen Sie dann den Mauscursor in das Feld OK und klicken Sie einmal mit der linken Maustaste. Vergleichen Sie dann bitte die Eingaben im LIST-Window nochmals genau mit unserem Programm und verbessern Sie eventuelle Tippfehler.

So, das war doch schon was, oder?

Wahrscheinlich haben Sie vieles von dem, was Sie eintippten, noch nicht verstanden. Aber das macht überhaupt nichts. Es sollte ja nur eine Kostprobe sein. Sie sehen hier, was mit recht

geringem Aufwand erreicht werden kann. Die einzelnen Befehle und Funktionen werden Sie im Grafik-Kapitel alle erklärt bekommen.

Wählen Sie jetzt bitte die Option "Save As" aus dem "Project"-Menü. Dazu drücken Sie die Menütaste, also die rechte Maustaste, und fahren Sie mit dem Mauscursor auf das dann erschienene Wort "Project" in der Kopfleiste. Ziehen Sie den Mauszeiger im entstandenen Pulldown hinunter auf das Wort "Save As", bis diese Option schwarz hinterlegt ist. Lassen Sie dann die Menütaste los. In der linken oberen Ecke des Bildschirms erscheint ein Kästchen. Bewegen Sie den Mauscursor in dieses Kästchen, klicken Sie einmal mit der linken Maustaste und tippen Sie auf der Tastatur:

```
Ballprogramm <RETURN>
```

Das Kästchen wird wieder verschwinden, und das Diskettenlaufwerk des Amiga wird kurz arbeiten. Bitte warten Sie, bis die rote Lampe am Laufwerk wieder erloschen ist. Wir haben unser kleines Ballprogramm jetzt für zukünftige Experimente auf Diskette abgespeichert. Das ist wichtig: Ein BASIC-Programm ist ab dem Augenblick auf Nimmerwiedersehen verloren, in dem der Amiga ausgeschaltet wird. Egal, ob Sie das mit Absicht machen oder Ihre Katze bei der Jagd nach einer Fliege den Stecker aus der Dose reißt. Nur auf einer Diskette kann das Programm für die Zukunft aufgehoben werden.

Klicken Sie nun bitte ins BASIC-Window und geben Sie ein:

```
new <RETURN>
```

Sie sehen, daß der Inhalt von beiden Windows gelöscht wurde und der BASIC-Cursor nun der Dinge harrt, die da kommen mögen.

1.5 Mal hier, mal da... - BASIC-Window und LIST-Window

Was passiert überhaupt im LIST-Window mit unseren Eingaben, wenn sie nicht direkt ausgeführt werden? Oder - anders gefragt - welche unterschiedlichen Funktionen haben eigentlich BASIC-Window und LIST-Window? Unsere Experimente im BASIC-Window fanden im sogenannten Direktmodus statt. Der Name kommt daher, daß der Amiga die Eingaben, die hier stattfinden, direkt nach dem <RETURN> ausführt. Im LIST-Window befinden wir uns im Gegensatz dazu in einem Programmier-Modus. Hier merkt sich der Amiga die Eingaben solange, bis das Programm mit RUN gestartet wird.

Vielleicht haben Sie sich aber schon einmal abgedruckte Programme - in Zeitschriften oder Büchern werden sie Listings genannt - angesehen. Dann sind Ihnen sicher zuerst die vielen Zeilennummern aufgefallen, die jeweils vor den BASIC-Befehlen stehen. Wo sind also unsere Zeilennummern? Nein, wir wollen Ihnen nichts unterschlagen! Die Erklärung ist einfach: Es gibt sie nicht. Sie sind beim Amiga überflüssig. Herkömmliche Computer brauchen sie, und zwar aus mehreren Gründen. Einer davon ist, um zwischen Eingaben im Direktmodus und Programmiermodus zu unterscheiden.

Normalerweise signalisiert ein Befehl mit Zeilennummer dem Computer, daß das Folgende nicht sofort ausgeführt werden soll: Programmiermodus. Ein Befehl ohne Zeilennummer heißt für den Computer: Direktmodus. Für diese Unterscheidung gibt es aber beim Amiga ja schon die beiden Windows. Das heißt aber nicht, daß Sie bei AmigaBASIC keine Zeilennummern verwenden dürfen. Wenn Sie unbedingt möchten, geht das auch. Doch zu diesem Thema kommen wir etwas später.

Nach unserem kleinen Programm von vorhin ist es nun an der Zeit, etwas Umfangreicheres anzugehen. Fangen wir am besten mit unserem Videotitel-Programm an. Bitte tippen Sie alles folgende auch wirklich ab. Zum Schluß ergibt sich daraus nämlich ein Programm. Und fast nebenbei lernen Sie, mit Ihrem Amiga in einer der beliebtesten Computersprachen zu sprechen: BASIC.

Klicken Sie also bitte ins LIST-Window und geben Sie dort ein:

```
? "Videotitel-Programm" <RETURN>
? "von Franz Maier" <RETURN>
```

Damit sind die ersten Zeilen unseres Videoprogrammes auch schon programmiert. Und eigentlich müßten Sie auch alles, was da steht, schon verstehen. Wir haben bis jetzt programmiert, daß der Amiga den Titel unseres zukünftigen Programms und Ihren Namen auf dem Bildschirm darstellt. (Bei Franz Maier sollten Sie wieder Ihren Namen einsetzen.)

Wenn Sie jetzt überprüfen wollen, ob das Programm auch funktioniert: Der BASIC-Befehl, der Programme startet, heißt RUN. Diesen Befehl dürfen Sie in diesem Fall aber nicht einfach ins LIST-Window tippen, da sich der Amiga sonst erst mal gar nicht darum kümmern würde. Bei RUN handelt sich um einen Befehl, der meist im Direktmodus eingegeben wird. Also aktivieren wir das BASIC-Window und tippen:

```
run <RETURN>
```

Wenn Sie alles so eingegeben haben wie besprochen, dann ist nun folgendes passiert: Das LIST-Window ist vom Bildschirm verschwunden, im BASIC-Window ist unser Text erschienen, darunter steht OK und der Cursor. Die Ausgaben eines BASIC-Programms (was ein Programm druckt, nennt man eine Ausgabe) erscheinen also auch im BASIC-Window. Und wohin ist mein Programm verschwunden? Denken Sie daran, daß AmigaBASIC mit Windows arbeitet. Und Windows können bekanntlich hintereinander liegen. Genau das ist auch passiert. Jetzt gibt es mehrere Möglichkeiten, wieder an das Listing heranzukommen. Eine ist:

```
list <RETURN>
```

Wenn Sie diesen Befehl im BASIC-Window eintippen, erscheint das LIST-Window wieder.

Für alle, denen das jetzt schon wieder zu viel Tipperei war, haben die Amiga-Programmierer einige Hilfen eingebaut: Wie wir es von der Workbench schon kennen, gibt es auch bei Amiga-BASIC Pulldown-Menüs. Wenn Sie die rechte Maustaste drücken, sehen Sie, was da zur Auswahl steht. Es gibt vier Menütitel:

```
Project   Edit   Run   Windows
```

Das "Project"-Menü dient zur Verwaltung der BASIC-Programme. Hier gibt es Optionen zum Löschen, Laden und Speichern der Programme. Eine dieser Funktionen haben wir vorhin beim Abspeichern unseres Ballprogrammes schon einmal benutzt, erinnern Sie sich? Wir werden später noch genauer auf dieses Pulldown eingehen.

Das "Edit"-Menü bietet Hilfen beim Editieren, sprich Verbessern und Ändern von Programmen.

Das "Run"-Menü soll uns gleich näher interessieren. Hier gibt es verschiedene Optionen, um Programme zu starten, anzuhalten und während der Ausführung zu beeinflussen.

Das "Windows"-Menü schließlich hilft uns beim Umgang mit dem LIST-Window und dem BASIC-Window.

Um unser Programm zu starten, können wir auch die Option "Start" aus dem "Run"-Menü anwählen. Sie wissen ja noch: Rechte Maustaste gedrückt halten, mit dem Mauscursor auf "Run" fahren, dann bis "Start" im Pulldown herunterziehen und loslassen.

Und schon ist das Programm wieder gelaufen - und das Listing weg. Weil Sie aber das Listing sehen wollten, können Sie jetzt im "Windows"-Menü die Option "Show List" auswählen. Sie tut das, was der Name schon vermuten läßt: Sie zeigt das LIST-Window an. Umgekehrt zeigt "Show Output" das Ausgabe-Window, also in unserem Fall das BASIC-Window.

Vielleicht ist Ihnen aufgefallen, daß in den Pulldowns neben den Optionen "Start" und "Show List" noch etwas anderes zu sehen ist: Ein blau hinterlegtes A und daneben ein Buchstabe. Bei "Start" ein R und bei "Show List" ein L. Diese Kürzel weisen auf eine dritte Möglichkeit hin, Programme zu starten oder das LIST-Window anzuzeigen. Zwischen der Leertaste - dem länglichen Barren ganz unten an der Tastatur - und der rechten <ALT>-Taste sehen Sie eine Taste mit einem roten, nicht ausgefüllten A. Diese Taste ist eine der beiden Amiga-Tasten. Wir wollen Sie in Zukunft rechte oder "offene" Amiga-Taste nennen, wogegen die andere Taste mit dem ausgefüllten roten A die linke oder "geschlossene" Amiga-Taste ist.

Wenn Sie also die offene Amiga-Taste gedrückt halten und dann die Taste <R> drücken, erreichen Sie dasselbe wie durch Auswahl der "Start"-Option aus dem "Run"-Menü. Entsprechend bringt die Kombination <offene Amiga-Taste> und <L> das LIST-Window genauso auf den Bildschirm wie die "Show List"-Option aus dem "Windows"-Menü. Sie werden bei Ihrer zukünftigen Arbeit am Amiga noch weitere solcher Tasten-Kürzel kennenlernen. Gerade bei Programmen, die vorwiegend mit der Tastatur und weniger mit der Maus bedient werden, können sie sehr nützlich sein.

Welche der vorgestellten Möglichkeiten Sie verwenden, wenn es in Zukunft heißt, "Starten Sie bitte das Programm" oder "Holen Sie das LIST-Window auf den Bildschirm", bleibt ganz Ihnen überlassen.

1.6 Es geht voran - weitere BASIC-Funktionen

Nachdem wir nun schon ganz gut mit den beiden Windows und dem Starten von Programmen zurechtkommen, wäre es eigentlich an der Zeit, wieder ein wenig für unser Videotitel-Programm zu tun. Geben Sie bitte im LIST-Window die folgenden Zeilen ein:

```
? "Auswahl:" <RETURN>
? "1  Text eingeben" <RETURN>
? "2  Objects einlesen" <RETURN>
```

```
? "3  Obj.Bewegung festlegen" <RETURN>
? "4  Farben festlegen" <RETURN>
? "5  Titel wiedergeben" <RETURN>
```

Sechs neue Zeilen Programm - und Sie verstehen sie alle. Das zeigt zwei Dinge: Erstens, daß BASIC doch nicht so schwer ist, wie Sie vielleicht gedacht hatten, und zweitens, daß der PRINT Befehl ziemlich häufig verwendet wird. Mit ihm haben wir jetzt eine Art Auswahl-Menü erstellt. Gleichzeitig sehen Sie auch schon, was unser Programm zum Schluß können soll: Man soll Texte bzw. einen Titel eingeben können. Dann gibt es einen Punkt, um Objekte einzulesen, und einen, um ihre Bewegung festzulegen. Ein wenig Farbe könnte auch nicht schaden, deshalb der Programmpunkt 4, und zu guter Letzt brauchen wir einen Programmteil, der uns dann alles vorspielt, damit wir unseren Titel auch ansehen und später eventuell auf Video aufzeichnen können.

Wie sieht das alles jetzt auf dem Bildschirm aus? Am besten starten wir unser Programmfragment gleich mal. Dazu noch ein Tip: Wenn Sie das LIST-Window aktiviert lassen (also nicht ins BASIC-Window klicken) und dann "Start" aus dem "Run"-Menü wählen bzw. <offene Amiga-Taste> und <R> drücken, erscheint das Programmlisting nach dem Programmlauf automatisch wieder auf dem Schirm. Falls Sie das nicht wollen, einfach vorher das BASIC-Window aktivieren.

Starten Sie das Programm.

Naja, sieht ja ganz hübsch aus, hängt aber vielleicht ein wenig eng aufeinander. Da müßte sich doch etwas machen lassen. Holen Sie bitte das LIST-Window auf dem Bildschirm und aktivieren Sie es. Bewegen Sie nun den BASIC-Cursor hinter das Anführungszeichen am Ende Ihres Namens. Wenn er dort steht, drücken Sie bitte <RETURN> - schon ist eine freie Zeile entstanden. So einfach schafft man sich mehr Platz.

Bitte geben Sie an diese Stelle ein:

```
?
```

Also einfach einen PRINT-Befehl, ohne <RETURN> oder irgendetwas anderes. Drücken Sie dann die Cursor-Taste, auf der der Pfeil nach unten zeigt. Wieder wurde aus dem ? ein PRINT.

Wenn Sie das Programm starten, werden Sie sehen, daß zwischen Ihrem Namen und dem Wort "Auswahl:" eine Leerzeile ausgegeben wird. Ein PRINT-Befehl ohne einen Text oder eine Variable dahinter erzeugt also eine Leerzeile.

Es gibt auch eine Möglichkeit, die das genaue Gegenteil bewirkt: Bewegen Sie den BASIC-Cursor im LIST-Window hinter das Wort "Videotitel-Programm". Schreiben Sie jetzt bitte hinter das Anführungszeichen einen Strichpunkt (;). Die ersten beiden Zeilen sehen dann so aus:

```
PRINT "Videotitel-Programm";
PRINT "von Franz Maier"
```

Starten Sie das Programm bitte. Wenn Sie auf die erste Zeile achten, werden Sie sehen, daß nun der Programmtitel und Ihr Name in einer Zeile stehen. Ein Strichpunkt hinter einem PRINT-Befehl bewirkt, daß die nächste PRINT-Ausgabe nahtlos an die letzte angeschlossen wird. Ein kleiner Schönheitsfehler bleibt: Aus "Programm" und "von" ist ein Wort geworden. Das ist auch logisch, denn wir haben dem Amiga nirgends gesagt, daß er dazwischen eine Leerstelle drucken soll. Fügen Sie bitte im LIST-Window vor dem Anführungszeichen am Ende der ersten Zeile eine Leerstelle ein. Dazu müssen Sie den BASIC-Cursor nur zwischen das "m" von "Programm" und das Anführungszeichen setzen und dann die Leertaste drücken. Was immer Sie im LIST-Window tippen, wird automatisch an der Stelle eingefügt, wo der Cursor steht.

Auf dem Bildschirm stehen jetzt die Punkte, die wir in unserem Programm zur Auswahl haben. Als nächstes müssen wir dem Amiga mitteilen können, für welchen Punkt wir uns entschieden haben. Dazu geben wir im LIST-Window folgende Zeilen ein:

```
? <RETURN>
? "Ihre Wahl:"; <RETURN>
input a <RETURN>
```

Das einzig Neue ist der INPUT-Befehl. Lassen Sie das Programm probehalber einmal laufen. Dabei wird Ihnen auffallen, daß im BASIC-Window nun der Cursor hinter dem Text "Ihre Wahl:?" steht. Ja, das stimmt, aber wo kommt das Fragezeichen her? Das Fragezeichen erscheint automatisch, um anzuzeigen, daß der Computer auf eine Eingabe wartet. Das ist nämlich genau die Funktion von INPUT, der Benutzer wird nach etwas gefragt. Geben Sie bitte eine Zahl ein, gefolgt von <RETURN>. Sollte das nicht funktionieren und stattdessen der Bildschirm kurz aufblitzen, beachten Sie: Wenn Eingaben erfolgen sollen, muß das BASIC-Window aktiv sein.

Danach ist das Programm fertig, es erscheint ein OK im BASIC-Window.

INPUT ist ein Befehl, der es möglich macht, innerhalb eines Programmes Daten einzugeben, also mit dem Amiga zu sprechen. Aber was macht der Computer dann mit diesen Daten? Geben Sie bitte im BASIC-Window (nicht im LIST-Window!!!) ein:

```
? a <RETURN>
```

Als Antwort werden Sie Ihre Zahl erhalten. Der Befehl INPUT a weist also der Variablen 'a' den Wert zu, den Sie eingegeben haben. (Deshalb übrigens auch "input", engl.: eingeben).

Wenn Sie ein freundlicher Computer-Benutzer sind, haben Sie natürlich eine Zahl zwischen 1 und 5 eingetippt. Es soll aber auch weniger freundliche Leute geben, die Computerprogramme gern dazu bringen wollen, "abzustürzen" oder "sich aufzuhängen". Nein, nein, Sie haben nicht "Die Schreckensinsel des Dr. Frankenstein" gekauft. Dies ist ein ganz normales AmigaBASIC-Buch. Und "abstürzen" oder "sich aufhängen" sind Ausdrücke, die sich Programmierer einfallen ließen, um bestimmte prekäre Zustände zu beschreiben, in die Computer nun mal kommen können. Absturz heißt zum Beispiel, daß ein Computer inner-

halb einer Rechenoperation völlig durcheinander kommt. Der Amiga sendet dann beispielsweise als letztes Lebenszeichen einen dicken roten Kasten auf den Bildschirm, in dem steht "Guru Meditation" und dahinter ein Gewirr von Zahlen und Zeichen. Dieser Text hat dem Systemabsturz auch den Spitznamen "Guru" gebracht. Wenn also ein Amiga in der Nähe ist, und Sie hören einen Spruch wie "Jetzt macht er den Guru!", dann suchen Sie gar nicht erst nach einem der freundlichen, rötlich gekleideten Inder, die in einem Rolls Royce vorbeifahren. Sie müssen auch keine Angst haben, daß jeder, der Ihr Programm bedient, so einen Fehler produzieren kann, indem er statt einer Zahl zwischen 1 und 5 - wie erwartet - zum Beispiel 6 eingibt. Das Ergebnis wäre zwar eine Fehlermeldung, aber die ist bei weitem nicht so schlimm wie ein Guru. Trotzdem ist es bei der Programmerstellung immer ratsam, alle Fehler miteinzuplanen, die ein Benutzer - absichtlich oder unabsichtlich - machen könnte.

Die Aufgabe ist also, abzufragen, ob eine Zahl kleiner als 1 oder größer als 5 eingegeben wurde. Wenn ja, soll das Programm halt noch mal anfangen. Auf BASIC heißt das so:

```
if a<1 or a>5 then run <RETURN>
```

Hängen Sie diese Zeile bitte im LIST-Window an Ihr Programm an.

Mit dem BASIC-Befehl IF...THEN können Sie das Programm Entscheidungen treffen lassen, oder wie der Fachmann sagt: verzweigen lassen. Zwischen IF und THEN steht dabei eine beliebige Bedingung. Hinter THEN steht, was der Computer tun soll, wenn die Bedingung zutrifft. Also:

IF (Bedingung trifft zu) THEN (tu irgendwas)

In unserem Beispiel heißt die Bedingung (a<1 or a>5). Das OR bewirkt, daß die Bedingung vom Computer dann als zutreffend betrachtet wird, wenn entweder a<1 *oder* a>5 ist. Beides kann ja nie auf einmal eintreten, denn eine Zahl kann nicht kleiner als 1 und gleichzeitig größer als 5 sein. Das RUN hinter THEN läßt

das Programm neu anfangen, wenn eine falsche Zahl eingegeben wurde. Diesen Befehl kennen Sie ja schon aus dem BASIC-Window. Neu ist bloß, daß man ihn auch innerhalb eines Programmes anwenden kann. Ab jetzt können Sie nach Herzenslust Zahlen eingeben. Starten Sie das Programm und geben Sie zunächst Zahlen wie 0, 6 oder 10 ein. Der Amiga wird Sie solange fragen, bis Sie eine der erlaubten Zahlen eingeben. Auch wenn Sie ihm Tausende bieten - Computer sind unbestechlich, im Gegensatz zu manchen Programmierern. Er ist erst dann zufrieden, wenn die Zahl stimmt.

Nur so am Rande: Mit der Handvoll an BASIC-Befehlen, die Sie bis jetzt gelernt haben, könnten Sie zum Beispiel schon ein Programm schreiben, mit dem Kinder Rechnen üben könnten. Sie müßten nur die Aufgaben stellen, den Amiga das Ergebnis vorher ausrechnen und in einer Variablen ablegen lassen. Dann wartet der Amiga auf das Ergbnis, das die Kinder eintippen und vergleicht es mit seinem Ergebnis. Wenn das Ergebnis richtig ist, schreibt Amiga "STIMMT" auf den Bildschirm, wenn nicht "Leider Falsch." und startet das Programm neu. Das alles geht mit ein paar PRINT und IF...THEN Befehlen. Jetzt aber zurück zu unserem Videotitel-Programm.

Vielleicht wollen Sie ja einen etwas verschlafenen Benutzer, der eine falsche Zahl eingibt, drastischer auf seinen Fehler aufmerksam machen. Da bietet sich zum Beispiel der hübsche Piepser an, mit dem der Amiga sonst auch auf Fehler hinweist. Fügen Sie hinter dem THEN folgendes ein:

```
beep :
```

Das sieht dann im Listing so aus:

```
IF a<1 OR a>5 THEN BEEP : RUN
```

Jetzt piept's also, wenn ein Eingabefehler gemacht wurde. Und ein kurzes Bildschirmblitzen bekommen Sie auch noch gratis: Es gehört mit zum BEEP-Befehl dazu. Sollten Sie es vergessen ha-

ben: Der Doppelpunkt hinter dem BEEP dient dazu, verschiedene Befehle, die in einer Zeile vorkommen, voneinander zu trennen.

Bisher funktioniert ja alles ganz gut. Aber ist es in großen Programmen immer sinnvoll, das Programm beim kleinsten Fehler des Benutzers neu beginnen zu lassen? Stellen Sie sich mal ein Textverarbeitungsprogramm vor, das, wenn sich der Benutzer einmal vertippt und auf zwei Tasten gleichzeitig drückt, sofort den gesamten Text löscht und von vorne anfängt. Der zuständige Programmierer sollte dann wohl sein Haus mit Stacheldraht umzäunen und einen Wassergraben drumherum ausheben...

Es gibt jedoch Möglichkeiten, das Programm an andere Punkte als den Programmanfang zu schicken. Beispielsweise mit dem Befehl GOTO. Mit GOTO kann man das Programm an jede beliebige Stelle springen lassen. (Dieses "springen lassen" ist auch so ein beliebtes Bild bei Programmierern - ganz wörtlich darf man sich das nicht vorstellen.) Aber vorher brauchen wir eine Methode, dem GOTO-Befehl klarzumachen, wo er eigentlich hin soll. Bei den BASICs von manchen anderen Computern dienen dazu wieder die Zeilennummern. Aber wir haben Ihnen ja vorhin gesagt, daß die beim Amiga unnötig sind. Also muß es an ihrer Stelle noch irgendetwas anderes geben. Das Zauberwort heißt Labels, auf Deutsch: Sprungmarken. Es ist ganz einfach: Die Programmteile, die das Ziel von GOTO- oder anderen Sprungbefehlen sind, bekommen einen Namen. Holen Sie bitte das LIST-Window auf dem Bildschirm, bewegen Sie den Cursor direkt an den Anfang des INPUT-Befehls, schaffen Sie mit <RETURN> eine Zeile Platz und geben Sie dort ein:

```
abfrage:
```

Wichtig ist der Doppelpunkt hinter dem Wort. Er zeigt an, daß es sich hierbei um ein Label handelt. Wir hätten den Programmteil natürlich auch "Hallo:", "Trallalla:", "Hannes-Kleines-Nachtprogramm:" oder "Programmanfang:" nennen können. Alle Wörter, die keine BASIC-Befehle sind, sind erlaubt. Es ist aber sinnvoll, Programmteile immer nach ihrer Funktion zu benennen, damit man sich auch hinterher noch auskennt.

Aus dem RUN-Befehl in der letzten Zeile machen wir jetzt ein

```
goto abfrage
```

Das geht ganz einfach: BASIC-Cursor hinter RUN, dann dreimal <BACKSPACE> und den neuen Text tippen. Dabei kann es passieren, daß Sie über die rechte Seite des LIST-Windows hinausschreiben müssen. In diesem Fall wird der gesamte Inhalt des Windows nach links verschoben. Wenn Sie mit dem Cursor zurückgehen, kommt der alte Inhalt wieder zum Vorschein. Die einfachste Lösung ist, das LIST-Window auf dem Bildschirm zu verschieben und zu vergrößern. Das geht so, wie Sie es auf der Workbench auch von jedem anderen Window gewohnt sind.

Noch ein Nachtrag zum Thema Labels: Es sind alle Buchstaben und Zahlen erlaubt, nur Leerstellen dürfen nicht innerhalb eines Labels vorkommen. Satzzeichen sind teilweise mißverständlich, können zu Fehlern führen (z.B. das Komma oder der Doppelpunkt) und sollten deshalb vermieden werden. Labels dürfen bis zu 40 Zeichen lang sein. Alles, was darüber hinausgeht, wird vom Amiga ignoriert. Das gilt übrigens auch für Variablen. Es muß also nicht immer 'a' sein, sondern es kann auch 'amiga' sein. Eine Anmerkung zur Schreibweise in unserem Buch: Die Namen von Variablen drucken wir in einfachen Anführungsstrichen, (z.B: ...die Variable 'Anfangswert'...), um sie eindeutig zu kennzeichnen. Diese Anführungsstriche dürfen Sie bei den Variablennamen nicht miteingeben.

Groß- und Kleinschreibung ist AmigaBASIC egal: Hallo, hallo und hALLO vertreten alle dasselbe Label bzw. dieselbe Variable. Im LIST-Window sorgt AmigaBASIC jedoch dafür, daß die Schreibweisen innerhalb des Programms einheitlich sind. Machen Sie folgendes Experiment: Geben Sie in der letzten Zeile

```
goto Abfrage
```

ein. Den Labelnamen haben wir diesmal mit einem Großbuchstaben am Anfang geschrieben. Drücken Sie nun <RETURN>

und schauen Sie gleichzeitig das Label zwei Zeilen darüber an (hier ist "abfrage" noch kleingeschrieben.)

AmigaBASIC übernimmt im Hinblick auf Groß-/Kleinschreibung die Schreibweise des zuletzt eingebenen Namens für alle gleichlautenden Namen im Programm. Aus "abfrage" wurde auch oben "Abfrage".

Wenn Sie wollen, können Sie jetzt weitere Labels einfügen. Außerdem ist es durch Einfügen von Leerzeilen möglich, Programmteile deutlicher voneinander abzugrenzen. AmigaBASIC unterstützt diese Möglichkeiten der Programmstrukturierung sehr stark. Bei uns sieht das Programm jetzt zum Beispiel so aus:

```
Anfang:
PRINT "Videotitel-Programm ";
PRINT "von Franz Maier"
PRINT

Auswahl:
PRINT "Auswahl:"
PRINT "1  Text eingeben"
PRINT "2  Objects einlesen"
PRINT "3  Obj.Bewegung festlegen"
PRINT "4  Farben festlegen"
PRINT "5  Titel wiedergeben"
PRINT
PRINT "Ihre Wahl:";

Abfrage:
INPUT a
IF a<1 OR a>5 THEN BEEP : GOTO Abfrage
```

Aber wir waren ja vorhin bei den Fehlern gewesen, die ein Anwender in unserem Programm machen kann. Einen davon haben Sie vielleicht auch schon gemacht, nämlich einen Buchstaben anstatt einer Zahl eingegeben. Dann wird Ihnen aufgefallen sein, daß der Amiga die Meldung "?Redo from Start" auf den Bildschirm brachte. Sie sehen, Amiga legt einen erstaunlichen Ein-

fallsreichtum an den Tag, wenn es darum geht, neue Fehlermeldungen zu produzieren. Was soll also "?Redo from Start" schon wieder heißen? Je nach Schul-Englisch kommt man auf eine Übersetzung wie "Noch mal von vorne anfangen." Das ist auch gemeint.

Mit Variablen hatten wir ja schon zu tun. Erinnern Sie sich an 'Hallo' oder 'a'? Wir haben diesen Variablen aber bisher immer Zahlen zugewiesen. Ist ja auch der Normalfall, oder haben Sie während Ihrer Schulzeit jemals in der Mathe-Stunde Lob geerntet, wenn Sie sagten: "Der Wert von y ist 'Hallo'!"? Aus diesem Grund ist es nicht möglich, einer Variablen wie 'a' oder 'Hallo' ein Wort oder einzelne Buchstaben zuzuweisen.

"Ach?! Was ist aber, wenn ich genau das tun will?" Kein Problem, dafür gibt es einen eigenen Variablentyp, die sogenannten Stringvariablen. String ist englisch und heißt "Kette", gemeint ist eine "Zeichenkette", also eine Kette aus einzelnen Zeichen und Buchstaben. Stringvariablen werden durch ein $-Zeichen gekennzeichnet. Der Variablen 'a$' oder 'Hallo$' kann man also beliebige Buchstabenkombinationen zuweisen. Geschieht eine solche Zuweisung innerhalb eines Programms, muß die Zeichenkette in Anführungszeichen geschrieben werden. Wenn wir die beiden letzten Zeilen des Programmteils 'Abfrage:' folgendermaßen ändern, sind wir auch gegen Buchstaben-Eingaben gewappnet:

```
INPUT a$
IF a$<"1" OR a$>"5" THEN BEEP : GOTO Abfrage
```

Sie fragen sich möglicherweise, warum wir soviel Zeit damit verbringen, das Programm gegen Fehleingaben abzusichern. Aber es ist wirklich besser, einen möglichen Fehler zuviel als einen zuwenig zu berücksichtigen: Wer ein Programm nicht kennt und bereits bei der ersten Eingabe einen Error präsentiert bekommt, der wird sicher kein allzugroßes Vertrauen mehr in dieses Programm haben. Und auch Sie werden sich beim Arbeiten ärgern, wenn ein kleiner Vertipper vielleicht das ganze Programm zum Abbruch bringt.

1.7 "Star Wars, Teil I" - Texteingabe im Videotitel-Programm

Ein anderer wichtiger Punkt, um ein Programm professionell erscheinen zu lassen, ist die Optik. Unsere Auswahl sieht ja schon ganz gut aus. Das einzige, was vielleicht noch stört, ist, daß nach einer Fehleingabe das Fragezeichen immer eine Zeile tiefer erscheint. Ein permanenter Fehlbediener hat es bereits nach einigen Eingaben geschafft, daß die Übersicht oben aus dem Bildschirm verschwindet, und dann weiß ja erst recht keiner, welche Eingaben erlaubt sind. Aus diesem Grund wollen wir noch ein letztes Mal an unserer Auswahl-Routine herumbasteln.

Damit wir uns die Arbeit dabei wieder so einfach wie möglich machen können, gibt es noch etwas über das LIST-Window zu lernen: Bisher haben wir immer die Cursor-Tasten benutzt, um den Cursor zu bewegen. Es geht aber auch anders, nämlich mit der Maus. Sie haben sicher schon einmal beim Aktivieren des LIST-Windows gemerkt, daß der BASIC-Cursor immer dort erscheint, wo Sie mit der Maus hinklicken. So ist es möglich, sich innerhalb des Programms schneller als mit den Cursor-Tasten zu bewegen.

Falls Sie aber die Maus versehentlich dabei verschieben und die rechte Maustaste gedrückt halten, entdecken Sie noch etwas völlig anderes: Vom BASIC-Cursor bis zum aktuellen Standpunkt des Mauscursors erscheint ein oranger Balken oder ein ganzes Feld, von dem der dort stehende Text hinterlegt wird. Gratulation, Sie haben soeben zufällig eine weitere Funktion des Bildschirm-Editors entdeckt. Jetzt wollen wir es mal mit Absicht machen.

Schreiben Sie bitte irgendwo im Programm einen beliebigen Text und fahren Sie mit der Maus dorthin. Positionieren Sie den BASIC-Cursor an den Anfang des Textes, halten Sie die rechte Maustaste gedrückt und bewegen Sie die Maus nun vorsichtig zum Ende dieses Textes. Passen Sie dabei auf, daß der Mauscursor in der richtigen Zeile bleibt, sonst werden darüber-

oder darunterliegende Zeilen auch noch hervorgehoben. Wenn genau der von Ihnen geschriebene Text hervorgehoben ist, lassen Sie die rechte Maustaste bitte los.

Geben Sie nun auf der Tastatur irgendeinen anderen Text ein. Sie sehen, wie der alte Text verschwindet und durch den neuen ersetzt wird. Heben Sie nun bitte auf dieselbe Weise den neuen Text hervor und drücken Sie dann die <BACKSPACE>-Taste. Der gesamte Text wird verschwinden. So ist es möglich, ganze Teile eines BASIC-Programms zu löschen. Sie sollten dabei aber verständlicherweise vorsichtig sein. Gelöscht ist so ein Stück Programm schnell, aber das Wiederherstellen ist sehr mühsam. Überzeugen Sie sich vor dem Löschen davon, daß nur der Text hervorgehoben ist, den Sie auch löschen wollten. Wenn Sie versehentlich zu viel löschen, ist bei dieser Methode der Text nämlich nicht mehr wiederzubekommen.

Doch Löschen ist noch lange nicht alles, was Sie mit dieser Funktion erreichen können. Heben Sie bitte den gesamten Abfrage-Teil mit der Maus hervor. Lassen Sie ihn hervorgehoben und wählen Sie das Pulldown-Menü "Edit" an. Aktivieren Sie nun bitte die Option "Cut". Wenn Sie nur ein Laufwerk besitzen, wird der Amiga Sie bitten, die Extras-Diskette zu entnehmen und die Workbench einzulegen. Um die Funktionen des "Edit"-Menüs ausführen zu können, braucht AmigaBASIC ein Programm, das sich auf der Workbench-Diskette befindet. (Es heißt übrigens "clipboard.device".) Wenn dieses Programm einmal geladen ist, können Sie die Extras-Diskette zurück ins Laufwerk schieben, da die Edit-Routinen nun im Speicher des Amiga stehen. Der Amiga merkt selbständig, welche Diskette im Laufwerk liegt. Darum brauchen Sie sich gar nicht weiter zu kümmern.

Wenn die Cut-Funktion ausgeführt ist, sehen Sie, daß... - der Abfrage-Teil verschwunden ist! Haben wir jetzt so einen Aufwand mit Diskettenwechseln und Pulldown-Funktion treiben müssen, nur damit ein Teil unseres schönen Programms verschwindet? Keine Sorge, das gute Stück ist noch da. Es befindet sich in einem speziellen Speicherbereich des Amiga, dem sogenannten Clipboard (deutsch: Notizbrett).

Wählen Sie nun - ohne die Position des BASIC-Cursors im LIST-Window zu verändern - "Paste" aus dem "Edit"-Menü. Schon ist der Text wieder da. Den Inhalt des Clipboards können Sie an derselben oder jeder anderen Stelle wiedereinsetzen. Und zwar beliebig oft. Wenn Sie das ausprobieren möchten, können Sie ja nochmal "Paste" anwählen. Schon sehen Sie den Auswahl-Teil doppelt - ohne eine Tropfen Alkohol. Jetzt können Sie einen der beiden Auswahl-Teile wieder mit "Cut" ausschneiden.

Die dritte Funktion dieses Menüs, "Copy", wirkt genauso wie "Cut", nur daß der aktivierte Teil nicht vom Bildschirm verschwindet, sondern ins Clipboard kopiert wird. Das ist besonders hilfreich, um Programm-Teile zu vervielfältigen.

Noch ein Trick zum Schluß: Um einzelne Wörter oder Ausdrücke zu aktivieren, genügt es, den Mauscursor an eine beliebige Stelle des Wortes zu positionieren und dann zweimal mit der linken Taste zu klicken.

Bitte bauen Sie mit diesem Wissen das Programm so um und erweitern es so, daß es folgendermaßen aussieht:

```
Anfang:
PRINT "Videotitel-Programm ";
PRINT "von Franz Maier"
PRINT

Auswahl:
PRINT "Auswahl:"
PRINT "1  Text eingeben"
PRINT "2  Objects einlesen"
PRINT "3  Obj.Bewegung festlegen"
PRINT "4  Farben festlegen"
PRINT "5  Titel wiedergeben"
PRINT

Abfrage:
LOCATE 10,1
PRINT "Ihre Wahl:";
INPUT a$
```

```
IF a$<"1" OR a$>"5" THEN BEEP : GOTO Abfrage
IF a$="1" THEN Texteingabe
PRINT "Punkt "a$" noch nicht vorhanden."
GOTO Abfrage
```

Was hat sich jetzt geändert? Solange unser Programm noch "im Bau" ist, soll darauf hingewiesen werden, wenn einzelne Punkte noch nicht vorhanden sind. Den Texteingabe-Teil wollen wir gleich schreiben, die anderen Eingaben (2 bis 5) führen zu der Nachricht "Punkt "a$" noch nicht vorhanden", die auf dem Bildschirm ausgegeben wird. Im String 'a$' steht ja die gewählte Nummer. Hier sehen Sie auch, wie man Text in Anführungsstrichen und Texte oder Zahlen von Variablen miteinander mischen kann.

Ein neuer Befehl ist auch noch dabei: LOCATE 10,1. Normalerweise werden die Ausgaben bei PRINT zeilenweise untereinander gedruckt. Mit LOCATE kann eine Position am Bildschirm (in unserem Fall die 10. Zeile und dort die 1. Zeichenposition) angegeben werden, ab der die nächste Ausgabe gedruckt wird. So haben wir es auf recht einfache Weise erreicht, daß der Text "Ihre Wahl:" bei Fehleingaben immer wieder in dieselbe Zeile gedruckt wird, anstatt nach einiger Zeit das ganze Auswahlmenü nach oben zu drängen. Sie können im Direktmodus (also im BASIC-Window) einige LOCATE-Befehle ausprobieren. Zum Beispiel:

```
locate 5,20 : ? "Hallo"
```

Doch zurück zu unseren Videotiteln: Wenn Sie das Programm starten und andere Werte als 1 eingeben, werden Sie sehen, was jeweils passiert. Jetzt sieht das ganze schon professioneller aus, und ziemlich narrensicher ist es auch. Nur bei der Eingabe von 1 gibt es noch eine Fehlermeldung. Kunststück, denn das Label 'Texteingabe:' auf das wir das Programm mittels GOTO springen lassen, existiert ja noch gar nicht. Das ist ungefähr so, als würden Sie jemanden zum Haus Nummer 15 in einer Straße schicken, in der es nur die Hausnummern 1 bis 4 gibt.

Noch etwas: Sollte das der erste Fehler sein, den Sie innerhalb eines BASIC-Programms fabriziert haben, beachten Sie bitte, daß AmigaBASIC das LIST-Window auf den Bildschirm bringt und die Zeile, in der der Fehler auftrat, hervorhebt. Durch Klicken ins OK-Feld der Fehlermeldung bestätigen Sie, daß Sie die Meldung gelesen haben. Um diesen Fehler zu umgehen, gibt es nur eine Lösung: Den fehlenden Texteingabe-Teil schreiben. Hängen Sie ihn bitte im LIST-Window an das bisherige Programm an:

```
Texteingabe:
CLS : INPUT "Wieviele Zeilen (1-15)";Zeilen$
IF Zeilen$="" THEN CLS : GOTO Anfang
Zeilen=VAL(Zeilen$)
IF Zeilen<1 OR Zeilen>15 THEN BEEP : GOTO Texteingabe
DIM Text$(Zeilen)
FOR x=1 TO Zeilen
LINE INPUT "Text:";Text$(x)
NEXT x : CLS : GOTO Anfang
```

Da gibt es jetzt wieder einiges an Neuigkeiten. Gehen wir es der Reihe nach durch. Ein Hinweis noch: Die folgenden kurzen Programme und Programmzeilen müssen Sie nicht abtippen - sie dienen nur der Demonstration der einzelnen Befehle und gehören nicht zum Videotitel-Programm.

Die Funktion VAL:

Wir kennen ja schon den Unterschied zwischen Zahlenvariablen vom Typ 'a' und Stringvariablen vom Typ 'a$'. Es kann in BASIC-Programmen notwendig werden, diese Variablentypen ineinander umzurechnen. Um aus einer Stringvariablen eine Zahlenvariable zu machen, gibt es die Funktion VAL. Das sieht dann so aus:

```
z = VAL (z$)
```

Wenn der String 'z$' als Text eine Zahl beinhaltet (z.B. "19"), nimmt 'z' den Wert 19 an. Beginnt der String aber mit einem

Buchstaben ("a123"), wird das Ergebnis 0. Folgen im String einer Zahl Buchstaben, wird der Wert bis zu dieser Stelle berechnet. Ein Beispiel: Die VAL("123a") ergibt 123, VAL("123a456") ergibt ebenfalls 123.

Der Befehl DIM:

Damit lernen wir einen dritten Variablentyp kennen, die sogenannten Felder. Felder werden verwendet, wenn viele verschiedene Daten unter einem gemeinsamen Namen gespeichert werden sollen. Vorher muß BASIC mitgeteilt werden, wieviele Elemente dieses Feld haben wird. Dazu dient der Befehl DIM.

Dazu ein Beispiel, daß leider etwas mathematisch sein muß: Sie haben mit DIM t$(10) ein Feld dimensioniert. Dieses Feld kann 11 verschiedene Elemente haben, nämlich t$(0), t$(1), t$(2), t$(3),... usw. bis t$(10). Und jedes dieser Elemente kann einen String beinhalten - also eine eigene Zeichenkette. Beachten Sie aber, daß das Feld 't$(z)' mit der "normalen" Variable 't$' überhaupt nichts zu tun hat.

Felder können auch mehrdimensional sein, z.B. DIM a$(2,2). Dann gibt es folgende 9 Elemente:

```
a$(0,0)  a$(0,1)  a$(0,2)
a$(1,0)  a$(1,1)  a$(1,2)
a$(2,0)  a$(2,1)  a$(2,2)
```

Und jedes davon beinhaltet wieder eine eigene Zeichenkette, die man frei zuweisen kann. Ein dreidimensionales Feld kann man sich als würfelförmiges Gitter vorstellen, an dessen Kreuzungspunkten sich jeweils ein String befindet. In AmigaBASIC können Felder bis zu 255 Dimensionen haben. Dafür läßt sich dann aber beim besten Willen kein praktisches Beispiel mehr finden. Wir werden später noch mehr mit Feldern zu tun haben.

FOR...NEXT:

Dieser wichtige Befehl dient dazu, den Wert einer Variable von einem Anfangswert bis zu einem Endwert hochzuzählen. Das folgende Beispiel druckt alle Zahlen von 1 bis 50:

```
for x=1 to 50
? x
next x
```

Beim ersten Durchlauf ist 'x'=1, das Programm trifft auf den NEXT-Befehl, kehrt zum zugehörigen FOR zurück, erhöht 'x' um 1, beim zweiten Durchlauf ist 'x'=2 usw. Ist der Endwert 50 erreicht, macht das Programm hinter dem NEXT-Befehl weiter. Aber weil da nichts mehr zum weitermachen ist, hört es sicherheitshalber mal auf.

LINE INPUT:

Bei einem normalen INPUT a$ gibt es einige Einschränkungen, so dürfen z.B. keine Anführungszeichen oder Kommas innerhalb des eingegebenen Texts vorkommen. Der LINE INPUT-Befehl erlaubt die Eingabe einer beliebigen Zeichenkette, ohne irgendwelche Einschränkungen. Alles, was bis zum Drücken der <RETURN>-Taste eingegeben wurde, wird dem angegebenen String zugewiesen.

Jetzt kennen wir die einzelnen Bestandteile des Texteingabe-Teils. Im Programm haben wir statt einzelner Buchstaben wie 'z' oder 't$' längere Variablennamen verwendet, die auf die Funktion der Variablen schließen lassen (z.B.: 'Text$', 'Zeilen'). Das ändert jedoch nichts an den verschiedenen Variablentypen und Funktionen, die wir gerade erklärt haben.

Wie funktioniert der Programmteil nun überhaupt? Zuerst fragt das Programm, wieviele Zeilen Text eingegeben werden sollen. Eine Zeile ist der Mindestwert, 15 die Höchstgrenze. Wenn dieser Wert bekannt ist, wird ein Stringfeld namens 'Text$(x)' mit der angegebenen Anzahl an Elementen dimensioniert. Mit so-

vielen LINE INPUT-Befehlen, wie Zeilen angegeben wurden, werden dann die Texte in dem Stringfeld 'Text$(x)' abgelegt. Von dort können sie später wieder abgerufen werden. Wenn alles fertig ist, löscht das Programm den Bildschirm und kehrt zur Auswahl zurück.

Auf diese Weise können Sie später Texte eingeben, die vor dem sich bewegenden Objekt erscheinen. Wenn Sie damit einen Videotitel produzieren wollen, würden Sie zum Beispiel hier den Titel des Filmes eingeben. Zum Beispiel einen Zweizeiler: "Star Wars" und "Teil I".

1.8 Safety first - Abspeichern

Mittlerweile ist unser Videotitel-Programm doch recht beträchtlich angewachsen. Doch noch ist es sehr vergänglich. Wenn nämlich der Amiga ausgeschaltet wird, egal ob durch verärgerte Eltern, gelangweile Ehepartner oder interessierte Kinder, dann wird das Programm nicht mehr wiederzubekommen sein. Es empfiehlt sich deswegen dringend, Programme schon während ihrer Entwicklung öfter auf Diskette abzuspeichern. Nach Vollendung eines Teils oder eines ganzen Programms ist es absolut notwendig.

Zu diesem Zwecke gibt es im "Project"-Pulldown zwei Optionen: "Save" und "Save As". Beide dienen dazu, das Programm, das gerade im Speicher steht, auf Diskette zu sichern. Worin unterscheiden sie sich dann? Wählen Sie bitte "Save As" aus. Auf Deutsch also "Speichere als...". Sollten Sie nur ein Laufwerk besitzen, und eine andere Diskette als die "ExtrasD" eingelegt haben, wird Sie der Amiga zunächst auffordern, die Diskette zu wechseln. Danach erscheint ein Kästchen im linken oberen Bildschirmeck, in dem steht: "Save program as:". Darunter befindet sich ein weiteres, leeres Kästchen. Bewegen Sie den Mauscursor bitte in dieses Feld und klicken Sie einmal. Damit haben Sie das Kästchen aktiviert, ein Text-Cursor erscheint.

Programme, die auf einer Diskette gespeichert werden sollen, brauchen einen Namen, damit sie der Amiga später auch wiederfinden kann. Unser Programm könnten wir sinnvollerweise "Videotitel" nennen. Tippen Sie diesen Namen in das Kästchen. Sollten Sie sich dabei vertippen, haben Sie diesmal gleich zwei Tasten, mit denen Sie löschen können: Die Taste <BACKSPACE> löscht von der Cursorposition nach links. Die Taste <DEL> löscht nach rechts. Das werden Sie zwar bei so kurzen Eingaben wie Programmnamen nicht unbedingt brauchen, aber dieses Prinzip ist bei Korrekturen längerer Eingaben sehr hilfreich. Wenn Sie den Namen eingetippt haben, klicken Sie bitte einmal ins Kästchen "OK". Das Kästchen verschwindet vom Bildschirm. Danach beginnt der Amiga, das Programm auf die Diskette zu schreiben.

Der Unterschied beim Punkt "Save" ist schnell erklärt: Wenn der Name des Programms einmal angegeben wurde, genügt es, die "Save"-Option zu wählen. Dann wird das Programm automatisch unter demselben Namen abgespeichert. Nur wenn sich der Name ändern soll, brauchen Sie wieder "Save As". Beim Abspeichern unter einem neuen Namen wird das alte Programm nicht gelöscht.

Wie so oft beim Amiga, gibt es noch eine Alternative zum Pulldown: Der Befehl SAVE kann auch direkt im BASIC-Window eingegeben werden. Dabei bewirkt

```
save <RETURN>
```

dasselbe, wie die gleichnamige Pulldown-Option;

```
save "Name" <RETURN>
```

entspricht der Option "Save As". Der Name des Programms muß dabei in Anführungszeichen angegeben werden.

Jetzt steht unser Programm auf Diskette. Wenn wir den Amiga jetzt ausschalten würden, könnten wir das Programm später wieder von der Diskette einlesen. Aber wer will schon jetzt, wenn's gerade spannend wird, ausschalten?

1.9 Alles neu macht das NEW - Löschen von BASIC-Programmen

Nachdem unser Programm jetzt sicher auf der Diskette liegt, können wir es im Speicher des Amiga erst mal löschen. Der Befehl, der BASIC-Programme löscht, heißt NEW. Sie können ihn wieder direkt im BASIC-Window eingeben oder die gleichnamige Option im "Project"-Pulldown wählen. AmigaBASIC löscht daraufhin beide Windows und bringt den Text-Cursor in das Window, das zuletzt aktiv war. Das Programm ist damit vollständig gelöscht, es befindet sich nicht mehr im internen Speicher des Amiga. (Als "internen" Speicher bezeichnet man die eingebauten Speicherbausteine. "Externe" Speicher sind Disketten, Festplatten etc.)

AmigaBASIC ist bereit für neue Eingaben. Sicher denken Sie jetzt: "Und was ist, wenn der Spanik und der Rügheimer mal vergessen, mir zu sagen, daß ich das Programm abspeichern soll, und dann gebe ich NEW ein..."

Ja, eigentlich wäre das Programm dann unwiederbringlich verloren. Aber Sie können es ja schon heraushören an unserem "eigentlich" und dem "wäre": Auch hier hat AmigaBASIC vorgesorgt. Lassen Sie uns einmal ein ganz kurzes BASIC-Programm im LIST-Window schreiben. Zum Beispiel:

```
FOR x=1 to 100
PRINT x
NEXT x
```

Nach den Erklärungen aus dem letzten Kapitel können Sie schon selbst erkennen, was dieses Programm tut. Es schreibt alle Zahlen von 1 bis 100 auf den Bildschirm. Das ist zwar nicht gerade atemberaubend, soll aber zur Demonstration genügen. Wenn Sie Lust haben, schauen Sie sich's vorher noch an. Jetzt versuchen Sie bitte, das Programm zu löschen: Entweder mit NEW im BASIC-Window oder im "Project"-Pulldown. Was hatten Sie erwartet? Daß das Programm einfach so verschwindet?

Nein, nein, der Amiga erlaubt uns nicht einfach, mit offenen Augen ins Unglück zu rennen. Statt den Speicher zu leeren, läßt der Amiga ein Kästchen auf dem Bildschirm erscheinen, in dem es heißt: "Current program is not saved. Do you want to save it before proceeding?" Also auf deutsch: "Das aktuelle Programm wurde nicht gespeichert. Wollen Sie es speichern, bevor Sie weitermachen?"

Dann gibt es drei verschiedene Kästchen, in die Sie klicken können: YES bedeutet: "Ja, ich möchte das Programm speichern." In diesem Fall erscheint das bekannte Kästchen, das uns fragt, wie das Programm heißen soll. NO bedeutet: "Nein, das Programm taugt eh' nichts. Ich will es nicht speichern." Erst jetzt wird die NEW-Funktion wirklich ausgeführt. CANCEL bedeutet: "Eigentlich wollte ich das Programm überhaupt nicht löschen, ich möchte lieber daran weiterarbeiten."

Das funktioniert aber nicht nur so, wenn wir ein völlig neues Programm eingegeben haben, sondern auch, wenn man an einem alten Programm etwas verändert hat. AmigaBASIC merkt, daß sich was getan hat. Und wenn Sie dann versuchen, ein NEW durchzuführen, stellt es Ihnen erstmal wieder die Gewissenfrage, wie oben beschrieben. Sie sehen: Der Amiga hilft uns wirklich, wo er nur kann. Unser kleines Demo-Programm können Sie jetzt übrigens wirklich löschen, wir brauchen es nicht mehr.

Die Methode NEW ist aber nicht die einzige Art, ein aktuelle BASIC-Programm in die ewigen Jagdgründe zu schicken. Es ist auch möglich, ganz aus AmigaBASIC auszusteigen und danach etwas völlig anderes zu tun. Auch dafür gibt es wieder verschiedene Möglichkeiten: Entweder Sie geben im Direktmodus den Befehl SYSTEM ein. Dieser Befehl beendet AmigaBASIC und bringt Sie zurück auf die Workbench. Oder Sie wählen die Option "Quit" im "Project"-Pulldown. Damit erreichen Sie dasselbe. Sie können schließlich, das ist die dritte Methode, das BASIC-Window und das LIST-Window jeweils mit dem Schließsymbol ausklicken: Auch dann verlassen Sie BASIC und kehren auf die Workbench-Oberfläche zurück. In allen Fällen würde Sie AmigaBASIC übrigens wieder daran erinnern, wenn Sie Ihr Programm noch nicht abgespeichert hätten.

Wie Sie es tun, bleibt Ihnen überlassen, aber bitte verlassen Sie jetzt AmigaBASIC für die Dauer eines Zwischenspiels.

Zwischenspiel 2: Jetzt wird abgeräumt - Anlegen einer Programm-Schublade

Und wieder einmal ist die Zeit reif für ein Zwischenspiel. In diesem hier wollen wir eine Schublade für unsere BASIC-Programme anlegen - und das hat natürlich nicht soviel mit Amiga-Grafik zu tun. Trotzdem ist es zum Weiterarbeiten notwendig. Jetzt aber von der Theorie zur Praxis. Das zweite Zwischenspiel kann beginnen.

Auf der Workbench wird jetzt im Vergleich zu dem geschäftigen Treiben auf dem BASIC-Bildschirm ziemliche Leere herrschen. Bestenfalls ist das Window der Extras-Diskette noch zu sehen. Sollte dies der Fall sein, schließen Sie bitte auch dieses Window durch Klicken in das Schließsymbol. Bitte öffnen Sie dann das Extras-Window wieder.

"Window zu, Window auf... - Was denn jetzt?" Wenn Sie das Gefühl haben, wir wissen nicht so recht, was wir wollen: Eine der Eigenheiten von Workbench ist, daß neue Icons erst dargestellt werden, wenn ein Window neu geöffnet wird. Beim zweiten Öffnen werden Sie jetzt acht Icons sehen: Die sechs, die wir schon kennen (Tools, FD1.2, Hinweise, AmigaBASIC, Trashcan und BasicDemos) sowie zwei weitere: Zum einen das Ballprogramm aus unserer Vorspeise und zum anderen ein Icon, das den Namen trägt, den Sie dem Videotitel-Programm gegeben haben. Um diese Icons vollständig zu sehen, müssen Sie das Extras-Window vielleicht etwas vergrößern. Sie sehen, die Icons sehen aus wie ein Blatt Papier aus einem Computerdrucker (deutlich zu erkennen an der "Lochung" am Rand des Papiers). Und die orangen Symbole darauf haben große Ähnlichkeit mit denen auf dem AmigaBASIC-Icon. So sehen Icons von BASIC-Programmen aus.

Falls Sie schon mal aus Interesse die Schublade mit dem Namen "BasicDemos" angeklickt haben, werden Sie dort bereits viele dieser Icons gesehen haben. Um die "BasicDemos"-Schublade werden wir uns im nächsten Kapitel etwas genauer kümmern. Dort sind eine Reihe sehr interessanter Beispielprogramme zu finden.

Wie Sie wissen, dient eine Schublade auf der Workbench dazu, Dinge aufzunehmen und so für ein gewisses Maß an Ordnung zu sorgen. Von Ordnung kann man bei uns aber nicht gerade sprechen. Immerhin liegen da einfach zwei einzelne Programme auf der sonst so aufgeräumten Benutzeroberfläche herum... Wir wollen uns deshalb eine eigene Schublade anlegen - für die Programme, die im Verlauf dieses Buches noch entstehen werden.

Achtung: Wir gehen davon aus, daß Sie mit einer Kopie Ihrer Extras-Diskette arbeiten. Auf die Original-Extras-Diskette sollten Sie keine eigenen Programme schreiben. Wie man Disketten kopiert, steht im Amiga-Anwender-Handbuch, Kapitel 3.7. Um auch versehentlichen Schreiboperationen vorzubeugen, sollten Sie auf den Original-Disketten, die Sie mit Ihrem Amiga erhalten haben, den Schreibschutz-Schieber auf "Schreibschutz" stellen. Sie sehen in der rechten unteren Ecke auf der Unterseite einer Diskette einen kleinen Plastik-Schieber. Dieser Schieber kann zwei Stellungen haben: Wenn er so steht, daß das kleine viereckige Loch in der Diskette geschlossen ist, können Sie auf dieser Diskette Programme und Daten abspeichern. Wenn Sie das Plastik-Stückchen so verschieben, daß das kleine Loch frei wird, ist die Diskette schreibgeschützt.

Öffnen Sie jetzt bitte die Workbench. Besitzer eines einzigen Laufwerks müssen dazu die Extras-Diskette entnehmen und die Workbench einlegen. Lassen Sie das Extras-Window aber bitte unbedingt offen! Sie werden im Workbench-Window eine Schublade namens "Empty" entdecken. Diese Leerschublade ist

extra dazu gedacht, von ihr Kopien zu machen. Aktivieren Sie also bitte die Schublade "Empty". Wählen Sie dann aus dem Pulldown "Workbench" die Option "Duplicate". Damit duplizieren Sie die Schublade. Nach wenigen Augenblicken wird im Workbench-Window eine neue Schublade namens "copy of Empty" entstehen. Verschieben Sie bitte die Windows so, daß beide (Extras und Workbench) vollständig auf dem Bildschirm zu sehen sind.

Klicken Sie dann auf die neue Schublade und halten Sie die linke Maustaste gedrückt. Sie sehen, daß aus dem Mauscursor eine Kopie des Schubladen-Icons geworden ist. Sie nämlich gerade dabei, ein Objekt zu bewegen. Fahren Sie mit diesem Symbol ins Extras-Window und lassen Sie los. Nun wird die Schublade auf die Extras-Diskette kopiert. Das macht Amiga automatisch, und Sie müssen dabei eigentlich nichts mehr tun. Nur wer kein Zweitlaufwerk besitzt, muß dazu insgesamt dreimal zwischen den Disketten "ExtrasD" und "Workbench" wechseln. Der Amiga sagt Ihnen immer genau, welche Diskette er gerade haben will. Denken Sie dabei aber bitte unbedingt daran, zu warten, bis die rote Lampe am Laufwerk erloschen ist, auch wenn auf dem Bildschirm schon die Nachricht steht, daß Sie die andere Diskette einlegen sollen.

Alles fertig? Prima, dann schließen Sie bitte das Workbench-Window. Aktivieren Sie die neue Schublade und wählen Sie die Option "Rename" aus dem "Workbench"-Pulldown. In der Mitte des Bildschirms erscheint eine Zeile mit dem Inhalt "copy of Empty" und einem Textcursor. Bevor Sie hier tippen können, müssen Sie unter Umständen mit der Maus in diese Zeile klikken. Löschen Sie dann mit der <DEL>-Taste den alten Namen und taufen Sie die Schublade auf einen Namen, der Ihnen sinnvoll erscheint. (Wie wäre es zum Beispiel mit "Meine Programme"?) Schließen Sie die Eingabe mit <RETURN> ab.

Schon haben wir eine eigene Programmschublade. Jetzt können wir auch unser Videotitel-Icon und das Ballprogramm-Icon darin verschwinden lassen. Verschieben Sie dazu zunächst eines der Icons so, daß es über der Schublade liegen würde und lassen Sie dann die Maustaste los. Auf diese Weise ist es möglich, Icons

in Schubladen bzw. im Trashcan oder in anderen Disketten abzulegen. Um alles weitere kümmert sich der Amiga selbständig. Wenn der Mauscursor wieder seine gewohnte Form annimmt, wiederholen Sie den Vorgang bitte mit dem anderen Icon.

Durch unsere Arbeiten haben wir allerdings die Extras-Diskette ein bißchen in Unordnung gebracht. Räumen wir doch noch schnell auf, bevor wir zum nächsten Kapitel gehen. Dazu werden die Icons einfach so lange verschoben, bis alles einigermaßen ordentlich nebeneinander liegt. Damit ist es allerdings noch nicht ganz getan. Beim Thema Ordnunghalten ist der Amiga nämlich einer von der eher vergeßlichen Sorte. Man muß ihm erst ausdrücklich klarmachen, daß die gegenwärtige Anordnung endgültig ist.

Aktivieren Sie dazu alle Icons innerhalb des Extras-Windows. "Halt, Halt - geht gar nicht. Wenn ich ein Icon aktiviere, schalte ich doch damit das andere wieder ab." Stimmt, aber es gibt einen Trick, um genau das zu verhindern. Halten Sie dazu die <SHIFT>-Taste auf der Tastatur gedrückt und aktivieren Sie dann mit der Maus ein Icon nach dem anderen. Sie sehen nun etwas, was es eigentlich gar nicht geben dürfte: Alle Icons bleiben aktiviert. Jetzt müssen wir dem Amiga bloß noch mitteilen, daß er alle aktivierten Icons in Zukunft so und nicht anders anordnen soll, wenn er das Fenster öffnet. Das bewirkt die Option "Snapshot" im "Special"-Menü. Mit diesem Schnappschuß fotografiert der Amiga sozusagen die gegenwärtige Anordnung, schreibt sie auf die Diskette und merkt sie sich somit für die Zukunft. Unser Schreibtisch wird von solange auf die festgelegte Weise aufgeräumt bleiben, bis wir einen neuen Snapshot ausführen lassen oder neue Programme dazukommen.

1.10 Und es bewegt sich doch! - Bobs und Sprites

Wir können uns eigentlich kaum vorstellen, daß Sie nicht schon einmal aus Neugier in die Schublade "BasicDemos" geschaut haben. In dieser Schublade finden Sie nämlich eine ganze Menge interessanter Beispielprogramme, die allesamt in AmigaBASIC geschrieben sind. Es gibt da eine ganze Menge zu sehen und zu

hören. Wenn Ihre Experimente in dieser Richtung bei einigen Programmen bisher daran scheiterten, daß Sie nicht verstanden, wie sie funktionieren bzw. was sie tun, dürfen wir Sie in den Anhang verweisen. Genauer gesagt, in den Anhang C, wo wir die einzelnen Programme der BasicDemos-Schublade vorstellen.

Zunächst soll uns aber nur ein einziges Programm interessieren. Bitte öffnen Sie jetzt die Schublade "BasicDemos". Der Aufbau des Windows wird ein Stück dauern, weil sich hier eine große Anzahl verschiedener Programme tummelt. Bitte suchen Sie das Programm, das den Namen "ObjEdit" trägt. Auch wenn Sie es auf den ersten Blick nicht finden - es ist sicher da irgendwo unter all den anderen. Sobald Sie es gefunden haben, starten Sie es bitte. Das heißt, klicken Sie zweimal auf das Programm, genauso wie sie es auch von AmigaBASIC oder einem anderen Amiga-Programm gewohnt sind.

Das Laden wird einen Augenblick dauern. Das ist auch kein Wunder, denn der Amiga lädt nicht nur das Programm, sondern auch AmigaBASIC. Wenn Sie nämlich ein BASIC-Programm auf der Workbench anklicken, wird BASIC gleich automatisch mitgeladen. Das macht die ganze Angelegenheit sehr einfach. Auch Leute, die absolut nichts von AmigaBASIC verstehen, können so BASIC-Programme benutzen.

Mittlerweile müßte Ihr Amiga auch soweit sein:

```
1 eingeben, wenn Sie Sprites erstellen wollen.
0 eingeben, wenn Sie BOBs erstellen wollen >
```

sollten Sie jetzt auf dem Bildschirm lesen können, und dahinter sehen Sie den Cursor, der Ihnen Ihre Wünsche von den Tasten ablesen möchte. Ihr Amiga fragt also, ob Sie Sprites oder Bobs hätten. Sprites oder Bobs?

Beim Amiga gibt es Objekte, die sich unabhängig vom Bildschirm-Hintergrund bewegen können. Mit einem solchen Objekt haben Sie übrigens schon die ganze Zeit zu tun, ohne es richtig zu wissen: Dem Mauscursor nämlich. Der Mauscursor kann sich ja immer auf dem Bildschirm bewegen, egal, ob da nun ein

Workbench-Fenster, ein Fenster von AmigaBASIC oder irgendetwas anderes im Hintergrund liegt. Objekte, die sich unabhängig vom Hintergrund bewegen können, sind eine wichtige Voraussetzung für Computer-Animation.

Was heißt eigentlich "unabhängig vom Hintergrund"? Nun, bei früheren Computern gab es solche beweglichen Objekte nicht. Wenn auf dem Bildschirm eine Bewegung dargestellt werden sollte, mußte eine kleine Grafik an einer bestimmten Stelle gezeichnet werden, gelöscht werden, an einer leicht versetzten Stelle neu gezeichnet werden, wieder gelöscht werden und so weiter. Durch diesen Trick entstand dann wie bei einem Zeichentrickfilm die Illusion einer Bewegung.

Dieses Prinzip hat aber einen entscheidenden Nachteil: Vom Zeichnen und Löschen wird auch der Hintergrund betroffen, also der Rest des Bildschirms, der sich eigentlich nicht verändern soll. An der Stelle, wo einmal eine bewegte Grafik war, war plötzlich nichts mehr zu sehen - sozusagen nur ein schwarzes Loch. Stellen Sie sich mal vor, überall, wo der Mauscursor vorbeikommt, verschwindet der Bildschirminhalt. Bald würde der Amiga-Bildschirm ausssehen wie ein Emmentaler. Deshalb gab es lange Zeit nur ganz einfache Animationen.

Erinnern Sie sich zum Beispiel noch an einen der Veteranen des Computerspieles? Er hieß PONG oder bei uns "Tischtennis". Alles was man sehen konnte, war ein weißes Viereck, eine Mittellinie und zwei schmale Rechtecke links und rechts, die von den Spielern mit Drehknöpfen bewegt werden konnten. Immer wenn das kleine Viereck auf die eigene Seite kam, versuchte man, es zu treffen und so zurückzuschlagen, daß der Mitspieler es nicht mehr erwischte. Jaja, eine Mark pro Spiel gab damals der eine oder andere aus. (Nebenbei bemerkt: Es müssen viele "eine oder andere" gewesen sein, denn der Erfinder des Spieles, ein gewisser Mister Nolan Bushnell gründete mit diese Markstücken ein Unternehmen, das gerade wieder sehr erfolgreich ist: Atari...).

Aber zurück zum Thema: Bei der ersten Version von PONG gab es keinen Hintergrund, weil damals einfach kein billiger Rechner in der Lage war, schnell genug zu rechnen. "Warum schnell?

Was hat das damit zu tun?" werden Sie jetzt wissen wollen. Nun, Schnelligkeit war lange Zeit das überhaupt einzige Mittel, um Animation mit Hintergund möglich zu machen. Die Programmierer hatten sich nämlich folgende Lösung für das Problem ausgedacht: Objekte wurden bei Bewegungen nicht mehr nur einfach gelöscht, sondern die Programme so geschrieben, daß sie sich den Hintergrund merken und ihn nach Vorbeiziehen des bewegten Teils restaurieren. Darunter aber litt wieder die Geschwindigkeit, weil ja nun ein erhebliches Maß an Rechnerei nötig war. Computeranimation war also mit vernünftigem Aufwand auf kleineren Computern nicht zu realisieren.

Irgendwann haben sich die Ingenieure dann überlegt, daß es gar nicht schlecht wäre, einen Grafik-Chip zu konstruieren, der einem diesen ganzen Aufwand abnimmt. Solche Chips wurden entwickelt und unter anderem in die damals gerade aufkommenden Homecomputer eingebaut: Nun war es möglich, kleinere Grafikobjekte vor den normalen Bildschirminhalt einzublenden und zu bewegen, ohne sich weiter um den Hintergrund kümmern zu müssen. Bei den verschiedenen Geräten gab es dafür verschiedene Namen. Die bekanntesten sind: Player-Missile-Grafiken (z.B. bei den Atari 400/800-Computern) oder Sprites (z.B. beim Commodore 64).

Auch im Amiga finden sich solche Grafikobjekte. Hier gibt es sogar gleich zwei verschiedene Arten: Zum einen die Sprites (sie heißen nicht nur so wie beim C64, sie funktionieren größtenteils auch so) und zum anderen die Bobs, die Blitter-Object-Blocks. Was die Vor- und Nachteile jeder Art sind, werden wir später noch klären.

Und damit sind wir wieder beim Object Editor. Dieses Hilfsprogramm erlaubt Ihnen nämlich, bewegte Grafikobjekte zu definieren, und zwar sowohl Sprites als auch Bobs. Alles, was das Programm jetzt von Ihnen wissen möchte, ist, ob Sie einen Bob oder einen Sprite zeichnen möchten. Die Entscheidung nehmen wir Ihnen vorerst noch ab.

1.11 Wir machen einen Star - der Object Editor

Wir wollen ein wenig Bewegung in unser Videotitel-Programm bringen. Dazu wollen wir ein beliebiges Objekt, sagen wir einen Stern, über unseren Titel-Bildschirm fliegen lassen. Zu diesem Zweck verwenden wir einen Bob. Geben Sie jetzt bitte eine '0' ein, gefolgt von <RETURN>. Es wird nun ein Bildschirm erscheinen, auf dem Sie in erster Linie ein Art Window, eine Auswahl aus vier Farben und den Text "Bob-Größe X:31 Y:31 Stift" sehen.

Wenn Sie Schwierigkeiten damit haben, in dem Linien-Gebilde in der linken oberen Ecke des Bildschirms ein Window zu erkennen, dann klicken Sie bitte mit der Maus in das kleine Feld, wo bei einem "normalen" Window das Größen-Symbol ist. Bewegen Sie nun die Maus bei gedrückter Taste. Sie sehen, der Rahmen des Windows wird orange und bewegt sich. Wenn Sie loslassen, bleibt das Window in der neuen Größe stehen. Wahrscheinlich haben Sie es schon erraten: Innerhalb dieses Feldes kann unser Bob definiert werden.

Wenn Sie noch ein wenig mit der Größen-Funktion experimentieren, werden Sie feststellen, daß es horizontal und vertikal eine Begrenzung gibt, über die das Window nicht hinauskommt. In diesem Fall verschwindet der orange Rahmen, während Sie versuchen, ihn zu vergrößern. Wenn Sie sich außerdem in der unteren Zeile die Zahlen hinter X: und Y: ansehen, stellen Sie fest, daß sich diese Zahlen nach dem Verändern der Window-Größe ebenfalls ändern. Diese Zahlen geben die momentane Ausdehnung des Windows in horizontaler (X:) und vertikaler (Y:) Richtung an. Drücken Sie jetzt bitte die Menütaste der Maus. Am oberen Bildschirmrand sind drei Menüpunkte zu sehen:

```
File   Werkzeuge   Größe
```

Wir interessieren uns zunächst für das "Werkzeuge"-Menü.Nach Hammer, Meißel und ähnlichem werden Sie hier aber lang suchen müssen. Hier finden Sie stattdessen:

Stift

Dieser Modus ist nach dem Starten des Object Editors aktiv. Der jeweils aktive Modus steht am Ende der Zeile, wo auch die Window-Ausdehnung zu sehen ist. Im Stift-Modus können Sie freihandzeichnen: Der Object Editor funktioniert im Prinzip genauso wie die Zeichenprogramme Graphicraft oder DeluxePaint, die es für den Amiga gibt. Nur, daß dieses BASIC-Programm natürlich wesentlich einfacher und lange nicht so komfortabel ist. Wenn Sie im Stift-Modus den Mauscursor bei gedrückter linker Taste in dem stilisierten Window bewegen, sehen Sie, wie Sie mit der Spitze des Mauspfeils einen Strich zeichnen. So ist es mit ein wenig Übung und Geduld möglich, beliebige Figuren zu erstellen.

Linie

Da es im Stift-Modus sehr schwer ist, gerade Linien zu ziehen, gibt es diese Option. Sie klicken mit der Maus den Anfangspunkt der Linie an, halten die linke Taste gedrückt und bewegen den Mauspfeil zum Endpunkt der Linie. Anfangs- und Endpunkt werden bereits während des Verschiebens der Maus durch eine bewegliche orange Linie miteinander verbunden. So sehen Sie gleich, wie die Linie später verlaufen wird.

Oval

Mit Oval ist es möglich, Kreise oder Ellipsen zu zeichnen. Sie geben dabei mit der Maus die Eckpunkte eines Rechtecks an. Nach dem Loslassen der linken Maustaste wird innerhalb dieses Rechtecks der gewünschte Kreis gezeichnet.

Rechteck

Solange Sie die Maus bewegen, sieht diese Option genauso aus wie die Oval-Funktion. Aber in diesem Fall wird kein Kreis, sondern das dargestellte Rechteck gezeichnet.

Radierer

Da man sich ja mal verzeichnen kann, gibt es auch einen Radiergummi. Er radiert, solange die linke Maustaste gedrückt wird.

Fuellen

Damit ist es möglich, umrandete Flächen auszumalen. Zeigen Sie einfach mit dem Mauspfeil in die Fläche und klicken Sie einmal. Bei dieser Funktion ist jedoch Vorsicht geboten, weil man sich damit sehr schnell unbeabsichtigt Bilder zerstören kann. Zum einen müssen Sie beachten, daß der Fuellen-Befehl als Begrenzung einer auszumalenden Fläche nur Linien anerkennt, die in der gerade aktiven Farbe gezeichnet sind (Zur Farbauswahl kommen wir gleich). Zum anderen kann, wenn die Begrenzung auch nur an einem einzigen Bildpunkt offen ist, die Farbe "auslaufen", das heißt, sie malt auch Teile des Bildes außerhalb der angegebenen Begrenzung aus. Sicherheitshalber sollte man also beim Arbeiten mit diesem Befehl sehr häufig abspeichern.

Unterhalb des Windows sehen Sie vier Farbflächen (Blau, Weiß, Schwarz und Orange). Wenn Sie in eines dieser Kästchen klicken, können Sie die aktuelle Zeichenfarbe ändern. Das Wort "Farbe:" vor den Kästchen wird in der jeweils aktuellen Farbe geschrieben. Wenn Sie Blau als Zeichenfarbe wählen, verschwindet "Farbe:" natürlich, denn blaue Schrift auf blauem Grund ist nicht gerade gut lesbar. Wählen Sie zunächst Weiß als Zeichenfarbe.

Falls Sie unseren kleinen Durchgang durch das "Werkzeuge"-Menü mit einigen Experimenten verfolgt haben, müssen Sie jetzt

bitte zunächst Ihre so entstandene Zeichnung löschen. Wir nehmen nicht an, daß schon ein so tolles Kunstwerk herausgekommen ist, daß Sie es unbedingt aufheben wollen. Wählen Sie zum Löschen des Bildes die Option "New" aus dem "File"-Menü. Dann erscheint folgender Text auf dem Bildschirm:

> Aktuelle Daten sind nicht abgespeichert.
> Wollen Sie sie speichern?
> Y-Taste betätigen zum Speichern,
> N-Taste speichert nicht (Daten weg),
> C-Taste bricht Befehl ab.

Wenn Ihnen das jetzt irgendwie bekannt vorkommt, denken Sie mal dran, was BASIC macht, wenn Sie NEW eingeben und Ihre gegenwärtige Arbeit noch nicht abgespeichert haben. Der einzige Unterschied ist, daß diesmal die Tastatur und nicht die Maus zur Eingabe verwendet wird. Drücken Sie also bitte <N>, um Ihre Absicht zu bekräftigen, daß Sie nichts abspeichen wollen. Danach erscheint wieder die wohlbekannte Frage: Bobs oder Sprites? Wir bleiben bei Bobs, also geben Sie ein: 0 <RETURN>. (Falls Sie seit dem Start des Object Editors nichts gemalt haben, entfallen diese Dinge natürlich.)

Bitte vergrößern Sie das stilisierte Window jetzt so, daß X in der Gegend von 140 und Y in der Gegend von 70 liegt. Genau müssen Sie diese Werte nicht treffen. Das ist nur in etwa die Größe, in der wir unseren Stern programmieren wollen.

Wählen Sie dann die "Linie"-Option aus dem "Werkzeuge"-Menü. Mit diesen Linien zeichnen wir nun zunächst die Zacken unseres Sternes. Da wir den Stern später ausmalen wollen, achten Sie bitte schon jetzt darauf, daß keine Lücken entstehen. Das gelingt am besten, wenn Sie die Maus zwischen dem Ziehen der Linien gar nicht bewegen, so daß der Endpunkt einer Linie immer zugleich auch der Anfangspunkt der nächsten Linie ist. Wie das ungefähr aussehen sollte, zeigt unser Bild.

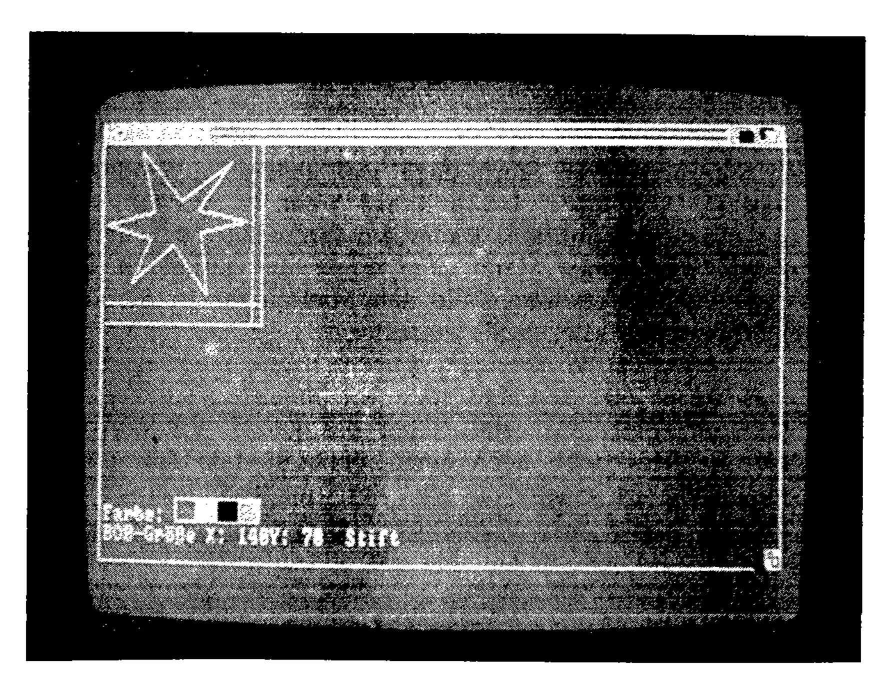


Bild 3: Der Object Editor

Wenn Sie mit dem Ergebnis zufrieden sind, wollen wir darangehen, den Stern in dieser Form abzuspeichern. Auch, wenn er noch nicht ganz fertig ist. Man sollte nämlich Zeichenprogrammen immer ein gewisses Maß an Mißtrauen entgegenbringen. (Und sich selbst gegenüber natürlich auch - denn wer macht schließlich keine Fehler? Eine kleine Unachtsamkeit und ein Kunstwerk ist der Welt genommen...)

Wählen Sie also "Save As" aus dem "File"-Menü. Auf dem Bildschirm heißt es nun "Dateinamen eingeben >". Taufen wir unseren Stern ruhig auf den Namen "Star", schließlich hat er noch eine große Zukunft vor sich. Schließen Sie den Namen mit <RETURN> ab. Jetzt wird die rote Laufwerkslampe aufleuchten: Der Bob wird gespeichert. Bitte machen Sie in dieser Zeit nichts mit dem Object Editor. Es ist zwar möglich, ihn auch

während des Speicherns zu bedienen, aber solange der Amiga mit Abspeichern beschäftigt ist, ist es sicherer, ihn in Ruhe zu lassen.

Schauen Sie jetzt Ihren Stern noch mal genau an: Finden Sie irgendwo ein Loch in der Umrandung? Wenn ja, stopfen Sie es bitte mit der Stift-Funktion. Dann aktivieren Sie die "Fuellen"-Option im "Werkzeuge"-Menü. Am Ende der untersten Zeile steht nun "Fuellen". Bewegen Sie den Mauscursor in die Mitte unseres Sternes und klicken Sie einmal.

Wenn der Stern jetzt hell und stolz vor Ihnen aufleuchtet, hat alles geklappt. Falls aber stattdessen das ganze Window weiß ausgefüllt ist, war doch noch eine Lücke in der Umrandung. Manchmal geht es einem da wie der Bundesregierung: Man weiß, irgendwo gibt es eine undichte Stelle, aber man findet sie einfach nicht... In diesem Fall hat sich unsere Vorsicht gelohnt. Wenn Sie jetzt nämlich "Open" aus dem "File"-Menü wählen, können Sie unseren "Star" wieder einlesen und erneut nach eventuellen Lecks suchen. Versuchen Sie's danach einfach nochmal mit dem Ausmalen.

Falls Sie versehentlich bei "Open" einen falschen Namen eingeben, wird der Object Editor mit einer Fehlermeldung abbrechen. Klicken Sie in diesem Fall ins OK-Feld des Errors und geben Sie RUN im BASIC-Window ein. Dann wird der Object Editor wieder gestartet.

Wenn beim Ausmalen alles geklappt hat, können wir unser Werk mit der "Save"-Option aus dem "File"-Menü abspeichern. Diese Option funktioniert ebenfalls genauso wie von BASIC gewohnt: Da wir schon einmal einen Dateinamen angegeben haben, wird jetzt automatisch unter demselben Namen abgespeichert.

Damit ist unser Star fertig. Wenn Sie jetzt noch ein wenig mit dem Object Editor spielen wollen: Eine Funktion haben wir bisher noch nicht erklärt. Im "Größe"-Menü gibt es die Option "4*4". Damit können Bobs vergrößert werden, allerdings nur, wenn Y kleiner als 31 und X kleiner als 100 ist. Bei größeren Bobs bringt der Amiga eine entsprechende Meldung. Sie müssen

diese Meldung durch einen Tastendruck bestätigen. Vergessen Sie das nicht, sonst kann der Object Editor sich manchmal recht seltsam verhalten. Im Größe-Modus ist nur die Stift-Funktion möglich. Mit der "1*1"-Option schalten Sie diesen Modus wieder aus.

Zum Verlassen des Object Editors klicken Sie bitte einfach ins Schließsymbol des BASIC-Windows. Falls während des Programmlaufs ein Fehler aufgetreten ist, müssen Sie unter Umständen auch noch das LIST-Window ausklicken.

Ja, und das war es auch schon.

Nein, das Buch ist natürlich noch nicht zu Ende - wir meinten, das war es auch schon, wenn es darum geht, ein Objekt zu definieren. So einfach geht das. Wo man bei anderen Computern mit vielen Tricks, PEEKs und POKEs und dergleichen arbeiten muß, genügt beim Amiga einfach Drauflosmalen. Schön, nicht wahr? Aber warten Sie erstmal ab, wie einfach es ist, beim Amiga etwas zu bewegen.

Wenn Sie den Object Editor verlassen haben, dann sehen Sie jetzt wieder die Workbench-Oberfläche. Bevor wir ins BASIC zurückkehren, sollten wir noch kurz für Ordnung sorgen. Dazu müssen Sie das BasicDemos-Window schließen und gleich danach wieder öffnen. Verschieben Sie nun bitte das sichtbar gewordene "Star"-Icon in die "Meine Programme"-Schublade (oder wie auch immer Sie die Schublade, die wir vorhin angelegt haben, genannt hatten). Sollten Sie beim Experimentieren weitere Bobs erzeugt haben, bringen Sie diese bitte auch in Ihre eigene Schublade. Dann können Sie das BasicDemos-Window endgültig zuklicken.

Wenn Sie wollen, räumen Sie mit der Snapshot-Funktion ruhig noch Ihre Programm-Schublade auf, Ordnung ist ja schließlich das halbe Leben. Die andere Hälfte sollte dafür vorwiegend Spaß machen. Und dafür wollen wir jetzt sorgen: Öffnen Sie bitte wieder AmigaBASIC.

1.12 Rollenverteilung - noch mehr über die Grafikobjekte

Wir haben ja nun schon mitbekommen, daß es zwei verschiedene Arten von Grafikobjekten auf dem Amiga gibt. Aber welche ist nun wofür am besten geeignet? Gibt es irgendwelche Vorteile bei der einen oder anderen Art? Oder umgekehrt irgendwelche Einschränkungen? Diese Fragen wollen wir jetzt beantworten.

Die Tatsache, daß es überhaupt zwei verschiedene Arten von Grafikobjekten im Amiga gibt, erklärt sich in der Entwicklungsgeschichte unseres Computers. Als im Jahre 1983 die Arbeiten daran begannen, hatte man noch im Sinn, einen Spielcomputer zu bauen, der alles bisher dagewesene in den Schatten stellen sollte. Die relativ kleine amerikanische Amiga Corporation engagierte einen Herrn namens Jay Minor, um die Spezialchips zu entwerfen. Dieser Jay Minor (er mußte wirklich mal erwähnt werden, denn ihm haben wir ja die meisten Möglichkeiten des Amiga zu verdanken) war just derjenige, der vorher schon die Grafikchips für die Atari 400/800-Serie entworfen hatte. Mit dieser Erfahrung war es ihm natürlich ein Leichtes, den Videochip "Denise" (so heißt er zumindest heute) so zu konstruieren, daß er neben den - im Vergleich zu den Atari-Chips - sehr viel leistungsfähigeren Farb- und Grafikmöglichkeiten fast nebenbei auch noch Sprites darstellen konnte.

"Denise" ist zwar am stärksten an der Bilderzeugung des Amiga beteiligt, aber alle Ehre gebührt nicht ihr allein. Es gibt da zum Beispiel auch noch den Chip "Agnus". Dieser Chip hat wiederum die beiden Unter-Baugruppen "Blitter" und "Copper", von denen Sie wahrscheinlich schon einmal gehört haben, weil es genau diese Chip-Funktionen waren, um die bei der Einführung des Amiga (nicht ganz zu Unrecht) ziemlich viel Wind gemacht wurde.

Die Hauptfunktion des Blitter ist es, Speicher- bzw. Bildbereiche blitzschnell zu kopieren oder zu bearbeiten. Dabei wird er vom Copper unterstützt, aber wie das genau funktioniert, brauchen Sie als BASIC-Programmierer wirklich nicht zu wissen. Dem Gespann Blitter und Copper haben wir übrigens auch die schnelle Verarbeitung von Windows zu verdanken. Daran können

Sie auch erkennen, daß solche Grafikchips eben nicht nur bei den reinen Grafikanwendungen Vorteile bieten, sondern auch sonst sehr nützlich sein können.

Und jetzt kommt etwas eigentlich sehr Kurioses: Durch den Blitter, den hochmodernen Superchip, wurde das älteste Konzept der Computergrafik für den Amiga wieder aktuell: Wenn es in schier unglaublicher Geschwindigkeit möglich ist, Bildbereiche zu verschieben und zu kopieren, warum sollte man dann nicht doch wieder zeichnen, verschieben und danach den Hintergrund restaurieren? So entstanden als Alternative bzw. Ergänzung zu den Sprites die Bobs, die Blitter Object Blocks. Grafikobjekte also, die durch den Blitter gesteuert werden.

Wenn man nun fragt, welche Objekte besser seien, muß man wissen: Da sie von völlig verschiedenen Chips erzeugt und gesteuert werden, unterscheiden sich Sprites und Bobs in verschiedenen Punkten wesentlich voneinander. Beide haben Vor- und Nachteile, abhängig von der jeweiligen Anwendung.

Über Sprites beim Amiga gibt es folgendes zu wissen: Sie sind in ihrer horizontalen Ausdehnung auf 16 Bildpunkte festgelegt. Das ist sehr schmal, es entspricht ungefähr zwei nebeneinanderliegenden Zeichen auf dem Bildschirm. Dafür sind sie in ihrer vertikalen Größe unbegrenzt. Sie sind aber auch in ihrer Farbauswahl eingeschränkt: Ein Sprite kann bis zu vier verschiedene Farben haben. Normalerweise gibt es 8 Sprites. Durch verschiedene Tricks ist es aber möglich, mehr als 8 Sprites gleichzeitig auf dem Bildschirm darzustellen. Außerdem bewegen sich Sprites schneller als Bobs.

Bobs können dagegen jede beliebige Größe annehmen, sowohl horizontal als auch vertikal. Sie können bis zu 32 verschiedene Farben haben, und die Anzahl der Bobs ist nur durch den zur Verfügung stehenen Speicherplatz begrenzt. Aber Bobs sind deutlich langsamer als Sprites, insbesondere dann, wenn viele Bobs gleichzeitig bewegt werden sollen. Die folgende Tabelle stellt die beiden Arten von Grafikobjekten noch einmal gegenüber.

Grafikobjekt:	Sprite	Bob
Gesteuert durch:	Denise	Agnus/Blitter
maximale Anzahl:	8 (aber mehrfach darstellbar)	unbegrenzt (soweit Speicherplatz ausreichend)
Farben:	bis zu 4 (für jeweils 2 Sprites gleiche Farben)	bis zu 32 (beliebig aus den aktuellen Farben auswählbar)
Geschwindigkeit:	sehr schnell	etwas langsamer

Tabelle 1: **Vergleich von Sprites und Bobs**

AmigaBASIC macht uns die ganze Sache bei der Programmierung sehr einfach: Für Sprites und Bobs werden dieselben Befehle verwendet - die Unterscheidung ist nur ein einziges Mal interessant, nämlich bei der Erzeugung des Objekts mit dem ObjectEditor.

Ein Nachtrag für uns deutsche Amiga-Anwender ist aber noch notwendig: Daß die Bewegung von Bobs flüssig und natürlich aussieht, ist in erster Linie eine Frage des Timings. Das Zeichnen und Verschieben muß innerhalb kürzester Zeit stattfinden, und zwar genau dann, wenn es der Anwender nicht sieht. Um dies zu erreichen, werden bestimmte technische Tricks verwendet.

Doch genau damit ergibt sich ein Problem: Der Amiga und AmigaBASIC sind in Amerika entwickelt worden. Dort gibt es andere technische Normen als in Deutschland. Für Fachleute und Interessierte: Die Netzfrequenz ist höher und die amerikanische Fernsehnorm benutzt weniger Bildzeilen als die deutsche. Die ersten Amigas, die es in Deutschland gab, waren amerikanische Geräte. Dort funktionierte auch die Bob-Darstellung bei jeder Bewegung perfekt. Die Amigas, die jetzt in Deutschland verkauft werden, wurden natürlich auf deutsche Verhältnisse ange-

paßt. Das bedeutet, sie laufen mit einer etwas niedrigeren Netzfrequenz und müssen mehr Zeilen auf dem Bildschirm darstellen. Dadurch hat es sich ergeben, daß sich Bobs nicht mehr so kontinuierlich wie gewohnt bewegen. Je größer ein Bob ist, bzw. je mehr Bobs verwendet werden, umso stärker beginnen diese Bobs zu flimmern. Dieses Problem konnte bisher nicht gelöst werden. Es ist aber zu erwarten, daß durch Modifikationen der Software (Kickstart, Workbench oder AmigaBASIC) der Flimmereffekt abgestellt werden kann. Zumindest arbeitet man bei Commodore und Microsoft daran. Bisher ist allerdings noch nicht bekannt, wann das Problem gelöst werden wird. Es kann also bei Bob-Animationen in BASIC passieren, daß die dargestellten Objekte während der Bewegung zittern.

Im Moment sollten Sie also noch nicht versuchen, "Krieg der Sterne" in AmigaBASIC zu programmieren. Dazu sollten Sie vorerst auf fertige Programme wie den Aegis Animator oder Deluxe Video zurückgreifen, bei denen diese Probleme nicht auftreten.

1.13 Ein Bob guckt in die Röhre - Einlesen von Grafikobjekten

Unser Stern ist ein Bob. Und zwar einfach deshalb, weil ein Sprite für einen ordentlichen Stern zu klein gewesen wäre. Mit unserem Bob haben wir jetzt ein Objekt, das wir bewegen können. Als nächstes wollen wir den Programmteil schreiben, in dem die Bewegung festgelegt wird. Bevor wir daran weiterarbeiten können, muß das Videotitel-Programm natürlich erst wieder in den Speicher geladen werden. Wählen Sie dazu bitte die Option "Open" aus dem "Project"-Pulldown.

In der linken oberen Ecke des Bildschirms erscheint nun ein Dialogfeld, das uns nach dem Namen des Programms fragt, das wir laden wollen. Klicken Sie bitte in das längliche leere Kästchen unterhalb des Satzes "Name of program to load:". Tippen Sie zuerst einen Doppelpunkt <:> und dann den Namen der Programmschublade, in der Sie das Videotitel-Programm und den

"Star" abgelegt haben. Drücken Sie dann die Taste </> und geben Sie danach den Namen des Video-Titelprogramms ein. Bei uns sieht das jetzt zum Beispiel so aus:

```
Meine Programme/Videotitel
```

Wenn Sie alles eingetippt haben, drücken Sie bitte <RETURN>. Falls der Amiga jetzt nicht lädt und stattdessen eine Fehlermeldung ausgibt (wahrscheinlich "File not found" - "Datei nicht gefunden"), haben Sie sich entweder beim Schubladen- oder beim Programmnamen geirrt. Sollten Sie sich nicht mehr an den richtigen Namen erinnern können, verkleinern Sie einfach das LIST- und das BASIC-Window und schauen Sie auf der Workbench nach. Sekunden nach Ihrer Eingabe erscheint das geladene Programm im LIST-Window.

Die Methode, wie wir unser Programm jetzt gerade geladen haben, ist die andere Möglichkeit neben dem direkten Anklicken von der Workbench aus. Nachdem es beim Speichern zwei Methoden gibt, können Sie sich wahrscheinlich schon denken, daß es neben der Pulldown-Option auch möglich ist, im Direktmodus einen Befehl zum Laden einzugeben. Er heißt LOAD und benötigt dieselben Angaben wie das Dialogfeld. Sie hätten also auch eingeben können:

```
LOAD "Meine Programme/Videotitel" <RETURN>
```

Was Sie lieber verwenden wollen, bleibt wieder Ihnen überlassen. Wenn Sie sich eher mit der Tastatur verbunden fühlen, ziehen Sie wahrscheinlich den gerade eben gezeigten Befehl vor. Wenn Sie eher ein Freund der Maus sind, dürfte das Pulldown das Richtige für Sie sein. Sie sehen, das läuft allmählich auf eine Grundsatzentscheidung heraus. AmigaBASIC unterstützt jedenfalls beide Möglichkeiten.

Wir wollen nun das zweite Unterprogramm schreiben. Das erste echte Unterprogramm war ja unser Texteingabeteil. Genauso programmieren wir jetzt das Einlesen des Grafikobjektes. Dazu nehmen wir zunächst den entsprechenden Punkt in unser Anfangsmenü mit auf. Bisher hieß es ja immer, wenn eine andere

Zahl als 1 eingegeben wurde, daß dieser Programmteil noch nicht existiert. Ab sofort soll es aber möglich sein, mit 2 die Einleseroutine zu aktivieren. Geben Sie also bitte im Programmteil 'Abfrage:' unter der Zeile

```
IF a$="1" THEN Texteingabe
```

die folgende Zeile ein:

```
IF a$="2" THEN ObjEinlesen
```

Drücken Sie dann <ALT><Pfeil nach unten>. Halten Sie also die <ALT>-Taste gedrückt und drücken Sie die unterste der vier Cursortasten: Der Cursor springt augenblicklich ans Programmende. Mit <ALT><Pfeil nach unten> gelangen Sie immer ans Ende des Programms. Entsprechend kommen Sie mit <ALT><Pfeil nach oben> an den Anfang. Und durch <ALT> in Verbindung mit einer der Tasten <Pfeil nach links> oder <Pfeil nach rechts> bringen Sie den Cursor an das linke bzw. rechte Ende der aktuellen Zeile. Diese Tastaturfunktionen sind eine weitere nützliche Hilfe, den Cursor schneller durchs Programm zu bewegen.

Am Ende des Programmes geben wir jetzt den nächsten Teil unseres Videotitel-Programms ein:

```
ObjEinlesen:
CLS
PRINT "Bitte geben Sie den Namen des Objekts ein."
INPUT Objname$
IF Objname$="" THEN CLS : GOTO Anfang
OPEN Objname$ FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
Eingel=1 : CLS : GOTO Anfang
```

Hier wird jetzt also ein Grafikobjekt von Diskette eingelesen. Die Neuigkeiten konzentrieren sich eigentlich auf drei Zeilen. In denen häufen sich allerdings auch die unbekannten Befehle:

```
OPEN ((Name des Objekts)) FOR INPUT AS 1
  OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
```

Die erste Zeile ist dafür zuständig, die angegebene Datei (so nennt man Daten auf Diskette, die kein Programm sind) zum Lesen zu öffnen. Warum das nötig ist und was dahinter steckt, erfahren Sie im zweiten großen Kapitel dieses Buches, wo es um Daten und Diskettenverwaltung geht. Wir wollen uns ja jetzt auf die Grafik konzentrieren. Was passiert also bei diesem Befehl? Der Inhalt einer Datei wird gelesen. Den Namen der Datei erfährt das Programm durch die Eingaberoutine in den ersten drei Zeilen dieses Programmteiles. Hier ist auch eine Sicherungsroutine eingebaut: Für den Fall, daß der Anwender aus Versehen <RETURN> drückt, ohne einen Namen einzugeben, erkennt unser Programm dies und springt an den Anfang zurück.

Jetzt werden die eingelesenen Daten einer Stringvariablen namens INPUT$ übergeben. INPUT$ ist ein String, genauso wie 'Hallo$' oder 'a$', nur mit dem Unterschied, daß Sie ihm den Inhalt einer Datei zuweisen können. Näheres dazu wieder im zweiten Teil unseres Buchs, dem Daten-Kapitel. Das ist ohne größere Probleme möglich, weil Strings in AmigaBASIC bis zu 32767 Zeichen lang sein dürfen. Dieser Platz reicht für die Daten eines Bobs oder eines Sprites im allgemeinen aus. In dem String stehen in genau festgelegter Form und Reihenfolge alle notwendigen Daten des Objekts, also beispielsweise unsere*s Sterns. Dank Object Editor brauchen wir uns um den Aufbau dieses Daten-Strings nicht zu kümmern, denn das hat Amiga schon erledigt.

In der zweiten der drei Zeilen wird der String mit dem Befehl OBJECT.SHAPE dem Grafikobjekt Nummer 1 zugewiesen. Durch diesen Befehl erhält AmigaBASIC alle benötigten Informationen über das Aussehen des Objekts, seine Farben, seine Höhe und Breite und einige andere Parameter. Die dritte Zeile schließt die zum Lesen geöffnete Datei wieder.

Dann wird noch eine Variable namens 'Eingel' auf 1 gesetzt: Sie dient im Programm dazu, herauszufinden, ob schon ein Objekt eingelesen wurde oder noch nicht. Warum es für das Programm wichtig ist, das festzustellen, werden Sie gleich verstehen, wenn wir zum nächsten Teil, der Bewegung, kommen. In jedem Fall kehrt das Programm nach dieser Formalität zum Hauptmenü zurück.

Sobald das Aussehen eines Objekts festgelegt ist, kann seine Bewegung programmiert werden. Fügen Sie dazu bitte zunächst im Programmteil 'Abfrage:' unterhalb der beiden schon vorhandenen Punkte die folgende Zeile ein:

```
IF a$="3" THEN ObjBewegung
```

Tippen Sie dann bitte am Ende des Listings den nächsten Teil ein:

```
ObjBewegung:
CLS : IF Eingel=0 THEN BEEP ELSE Bewegen
PRINT "Bitte erst Objekt einlesen!"
PRINT "Drücken Sie eine Taste!"
Warte:
a$=INKEY$
IF a$="" THEN Warte
CLS : GOTO Anfang
```

Wir kommen doch ganz gut voran, nicht wahr? Jetzt sind wir schon mitten im Programmieren. Und so schwer war es doch gar nicht. Zu den zuletzt eingetippten Zeilen: Dieser Teil ist der Beginn des Bewegungs-Programms und prüft, ob bereits ein Objekt eingelesen wurde. Sollte das nicht der Fall sein, haben wir ja nichts, was bewegt werden könnte. Dann soll der Amiga einen entsprechenden Hinweis auf den Bildschirm drucken, auf einen Tastendruck warten und danach zur Hauptauswahl zurückkehren.

Haben Sie vielleicht ein Problem mit der Zeile

PRINT "Drücken Sie eine Taste!"

und besteht dieses Problem vielleicht darin, daß sich AmigaBASIC hatnäckig weigert, das ü in "Drücken" auf dem Bildschirm stehen zu lassen? Falls Sie die letzte Frage mit einem klaren "Ja" beantworten können, dann haben Sie eine nicht ganz aktuelle Version von AmigaBASIC. Eine Zeitlang bestand das Problem, daß das kleine ü nicht in Programmlistings dargestellt werden konnte. Das Problem tritt nur bei dem kleinen ü auf, alle anderen deutschen Umlaute machen keine Schwierigkeiten. Sie wissen es ja mittlerweile: Fragen Sie Ihren Händler, ob er Ihnen die neueste Version von AmigaBASIC kopieren könnte. Und wenn der sie auch nicht hat, bitten Sie ihn, sich an Commodore zu wenden, um für seine Kunden die neueste Version zu besorgen. Sollte bei Ihnen mit dem kleinen ü alles funktionieren, dann freuen Sie sich, denn das ist ein Zeichen dafür, daß Sie wirklich eine der jüngsten Versionen von AmigaBASIC besitzen.

Zwei neue Befehle gibt es in diesem Programmteil: ELSE und INKEY$. ELSE ist eine Erweiterung des bereits bekannten IF...THEN. Bisher kennen wir nur die Form IF (Bedingung trifft zu) THEN (tu irgendwas). Zu dieser Grundform des IF...THEN-Befehls gibt es einige Erweiterungen, die Sie in diesem Buch Schritt für Schritt kennenlernen werden.

Schauen wir uns die betreffende Zeile im Programm mal genauer an:

```
IF Eingel=0 THEN BEEP ELSE Bewegen
```

Im vorderen Teil der Zeile funktioniert alles wie gehabt: Wenn die Variable 'Eingel' gleich 0 ist, dann soll der Amiga einen Piepser erzeugen. ELSE ist Englisch und heißt "anderenfalls". Damit läßt sich seine Funktion schon vermuten: Wenn 'Eingel' gleich 0 ist, piepst der Amiga, anderenfalls soll er zum Programmteil 'Bewegen:' springen. Durch IF...THEN...ELSE kann man dem Amiga zusätzlich mitteilen, was er tun soll, wenn die angegebene Bedingung nicht zutrifft. Das sieht dann so aus:

IF (Bedingung) THEN (tu irgendwas) ELSE (tu irgendwas anderes)

Um diesen Befehl in eine etwas alltäglichere Form zu bringen: Stellen Sie sich vor, ein Bekannter hat sich einen Amiga gekauft und will jetzt in AmigaBASIC programmieren. Er fragt Sie um Rat, woher er ein gutes Buch dazu bekommen soll. Sie sagen (so hoffen wir zumindest): "Wenn Du in einen Computerladen gehst, dann kauf' Dir das BASIC-Buch von Data Becker. Und wenn's nicht da ist, laß' es bestellen." In BASIC übersetzt hieße dieser Tip: "IF AmigaBASIC-Buch von Data Becker im Laden vorrätig THEN kaufe es ELSE bestelle es".

Diese Form von IF...THEN ist vor allem dann interessant, wenn - wie in unserem Fall - bei zutreffender Bedingung mehr als nur ein oder zwei Befehle abzuarbeiten sind und das Programm bei nicht zutreffender Bedingung an eine andere Stelle verzweigen soll. Das war hoffentlich nicht zu kompliziert? Sollten Sie es noch nicht ganz verstanden haben, werden Sie gleich eine Möglichkeit kennenlernen, wie Sie Sprünge und Schleifen besser nachvollziehen können.

Doch vorher noch ein letzter Befehl: INKEY$. Der ist wirklich ganz einfach: Dieser String beinhaltet immer das Zeichen, das gerade auf der Tastatur eingegeben wurde. Wenn keine Taste gedrückt wurde, ergibt INKEY$ einen leeren String. Deshalb brauchen wir auch die 'Warte:'-Schleife, die solange wartet, bis 'a$' einen anderen Wert als einen Leerstring ("") annimmt, weil Sie eine Taste gedrückt haben.

1.14 Amiga als Spurensucher - die Trace-Funktion

Falls Sie vor lauter Schleifen und Abfragen ein wenig die Orientierung verloren haben, hat der Amiga wieder eine nützliche Hilfe für Sie parat. Klicken Sie bitte ins BASIC-Window und tippen Sie:

```
goto ObjBewegung <RETURN>
```

Auf diese Weise ist es möglich, Programme direkt an einer bestimmten Stelle zu starten. Gleich piepst es, und der programmierte Text erscheint auf dem Bildschirm, denn ein Objekt wurde bisher ja sicher noch nicht eingelesen. Sollten Sie das LIST-Window nicht auf dem Bildschirm sehen, wählen Sie bitte "Show List" aus dem "Windows"-Pulldown. Dann wählen Sie die Option "Trace On" aus dem "Run"-Pulldown. Im LIST-Window erscheint jetzt das 'Warte:'-Programm, und ein kleiner oranger Kasten umrandet ziemlich flink einen Befehl nach dem anderen. Genauer gesagt wechselt er immer zwischen drei Zeilen.

Wir haben gerade den sogenannten Trace-Modus eingeschaltet. In diesem Modus zeigt uns der Amiga im LIST-Window, welcher Befehl gerade abgearbeitet wird. Davon, daß das eine große Hilfe beim Austesten von eigenen und Verstehen von fremden Programmen ist, können Sie sich gerade selbst überzeugen. Probieren Sie bitte aus, was passiert, wenn Sie jetzt eine Taste drücken. Denn darauf wartet der Amiga ja die ganze Zeit. Sie sehen, wie das Programm zum Anfang zurückspringt und während der Abarbeitung der einzelnen Befehle immer den orangen Kasten um den aktuellen Befehl legt.

Wenn auf einmal das LIST-Window verschwindet, zum Beispiel sobald der Amiga bei einem INPUT-Befehl ankommt, wählen Sie einfach wieder "Show List" aus dem "Windows"-Menü. Bei langen Programmzeilen oder Listings wird AmigaBASIC ziemlich häufig den dargestellten Programm-Ausschnitt im LIST-Window ändern. Vergrößern Sie im Bedarfsfall das LIST-Window nach Ihren Bedürfnissen. Wenn der Trace-Modus aktiv ist, müssen Sie einfach versuchen, die beste Plazierung von LIST- und BASIC-Window auf dem Schirm zu finden. Wie sie aussieht, hängt hauptsächlich davon ab, an welcher Stelle die Ausgaben im BASIC-Window erscheinen. Da sich unser Programm meistens in der oberen Hälfte des BASIC-Windows tummelt, haben wir bei unseren Versuchen das LIST-Window im unteren Teil des Bildschirms angesiedelt. AmigaBASIC paßt den dargestellten Ausschnitt des Listings immer der gegenwärtigen Größe des LIST-Windows an.

Wenn Sie wollen, können Sie jetzt mit aktiviertem Trace-Modus ein wenig im bisherigen Programm herumstöbern. Vielleicht werden dadurch die letzten eventuellen Unsicherheiten über die Funktion des Programms und seiner Teile beseitigt. Ihnen wird dabei auffallen, daß das Programm deutlich langsamer läuft als ohne den Trace-Modus. Das ist auch ganz logisch, denn zwischen zwei Befehlen hat der Amiga jetzt deutlich mehr zu tun als vorher. Der Trace-Modus ist ja auch nur zum Testen von Programmen gedacht.

Um ihn wieder abzuschalten, schauen Sie bitte ins "Run"-Pulldown. An der Stelle, wo vorher die Option "Trace On" stand, lesen Sie jetzt "Trace Off". Mit diesem Menüpunkt können Sie die Trace-Funktion wieder ausschalten.

AmigaBASIC bietet Ihnen auch die Möglichkeit, TRACE ganz gezielt für einzelne Programmteile zu verwenden. Es gibt die beiden Befehle TRON (TRace ON) und TROFF (TRace OFF), die Sie wie jeden anderen BASIC-Befehl im Programm verwenden können. Oder auch im Direktmodus anstelle der Pulldown-Optionen.

1.15 OBJECT hin, OBJECT her - die OBJECT-Befehle

Vielleicht ist Ihnen in der Zwischenzeit beim Experimentieren aufgefallen, daß im Texteingabe-Teil noch ein kleines Problem auftritt: Wenn man ihn nur einmal aufruft, klappt's prima. Aber beim zweiten Mal gibt es plötzlich eine "Duplicate Definition"-Fehlermeldung. Was ist passiert?

Das Problem ist der DIM-Befehl, der das Textfeld dimensionieren soll, sobald feststeht, wieviele Zeilen Text zu erwarten sind. Es ist nämlich nicht erlaubt, dasselbe Feld mit DIM mehrfach zu dimensionieren. Beim zweiten Versuch tritt der genannte Error auf. Das einfachste Mittel, um sich aus hier aus der Affäre zu ziehen, ist, alle Dimensionen mit DIM, die im Programm nötig sind, ganz vorne an den Programmanfang zu stellen. Dort werden sie dann einmal ausgeführt und später nicht mehr wiederholt.

Bei dieser Methode muß man allerdings wissen, wieviele Elemente das Feld im Höchstfall haben soll. Bei unserem Feld 'Text$' können bis zu 15 Elemente nötig werden. Wir legen also das Feld 'Text$' am Programmanfang einmal auf 15 Elemente fest. Das ist dann in jedem Fall ausreichend. Löschen Sie bitte zunächst den Befehl

```
DIM Text$(Zeilen)
```

im Texteingabe-Teil. Sie können ihn dazu mit Hilfe der Maus hervorheben und dann durch Drücken der Taste <BACKSPACE> löschen.

Man kann mit einem DIM-Befehl auch mehrere Felder gleichzeitig definieren. Wenn wir sowieso gerade dabei sind, erledigen wir deshalb gleich alle Felddimensionen, die im Verlauf unseres Programm noch benötigt werden. Wozu jedes einzelne Feld da ist, erklären wir jeweils an der Stelle, wo es benutzt wird. Bewegen Sie den Cursor jetzt an den Anfang des Listings, und geben Sie die folgenden Zeilen ein:

```
Vorbereitungen:
d=15
DIM Text$(d),Farbfeld(d,3),Beweg(d),Geschw(d)
```

Nachdem das letzte Stück Tippen ein bißchen kurz war, jetzt mal wieder etwas mehr. Übrigens - vergessen Sie bitte nicht, hie und da mal wieder das Programm abzuspeichern. Es ist ja seit dem letzten Mal ein beachtliches Stück angewachsen.

Übrigens: Wenn Sie sich den folgenden Teil jetzt ansehen und in Ihnen der Entschluß reift, das Listing lieber nicht abzutippen, dann wollen wir Sie an dieser Stelle nochmal an die Diskette im Buch erinnern. Sie finden das vollständige Videotitel-Programm in der Schublade "Videotitel". Der Programmname ist ebenfalls "Videotitel". Sollte es da beim Anklicken Probleme geben, dann denken Sie bitte daran, daß Sie zuerst "AmigaBASIC" auf die Diskette kopieren müssen. Wir haben Ihnen im Zwischenspiel 1 gezeigt, wie das geht.

Für alle, die aber doch lieber dem Prinzip "Lernen durch Tippen" folgen wollen, hier nun endlich das Listing:

```
Bewegen:
PRINT "Legen Sie bitte die Ausgangsposition fest."
PRINT "Bewegen Sie das Objekt mit den Cursortasten."
PRINT "Bestätigung mit <RETURN>"
ox=100 : oy=100 : Ziel=0
OBJECT.HIT 1,0,0
OBJECT.ON 1
OBJECT.STOP 1
Schleife:
a$=INKEY$
IF a$=CHR$(13) THEN Zieldef
IF a$=CHR$(28) THEN oy=oy-2
IF a$=CHR$(31) THEN ox=ox-5
IF a$=CHR$(30) THEN ox=ox+5
IF a$=CHR$(29) THEN oy=oy+[illegible]
OBJECT.X 1,ox : OBJECT[illegible] 1,o[illegible]
GOTO Schleife

Zieldef:
CLS
Beweg(Ziel*2+1)=[illegible]x : Beweg(Ziel*2+2)=oy
Ziel=Ziel+1 : Beweg(0)=Ziel
IF Ziel=7 THEN Enddef
PRINT "Bitte bewegen Sie das Objekt zum"Ziel". Zielpunkt."
PRINT "<RETURN> = Bestätigung"
PRINT "<ESC> = Ende"Schleife2:
a$=INKEY$
IF a$=CHR$(13) THEN Zieldef
IF a$=CHR$(27) THEN Enddef
IF a$=CHR$(28) THEN oy=oy-2
IF a$=CHR$(31) THEN ox=ox-5
IF a$=CHR$(30) THEN ox=ox+5
IF a$=CHR$(29) THEN oy=oy+2
OBJECT.X 1,ox : OBJECT.Y 1,oy
GOTO Schleife2
```

```
Enddef:
Beweg(0)=Ziel
OBJECT.OFF 1
CLS : GOTO Anfang
```

Na, auch wieder da? Sobald Sie dieses Listing abgetippt haben, haben Sie sich eine kleine Pause verdient. Weil es aus produktionstechnischen Gründen leider nicht möglich war, diesem Buch eine Packung Eis beizulegen, müssen wir Sie leider auf Ihren eigenen Kühlschrank verweisen. Guten Appetit! Die Pause sollten Sie auf jeden Fall auch zum Abspeichern benutzen.

Jetzt wieder frisch gestärkt ans Werk: Die Befehle, die wir anhand dieses Programmteils besprechen wollen, fallen in zwei unterschiedliche Gruppen: OBJECT-Befehle und andere.

Fangen wir mit den anderen an. Da gibt es nämlich nur einen einzigen Neuling: CHR$(x). Um ihn zu erklären, müssen wir einen kurzen theoretischen Abstecher einlegen. Lehnen Sie sich also am besten einen Moment zurück und machen Sie es sich bequem. Wir laden ein zur phantastischen Reise und zeigen Ihnen, wie das Gehirn Ihres Computers funktioniert.

Nach außen vermittelt unser Computer ja den Eindruck, als könne er mit Zahlen und Buchstaben ganz selbstverständlich umgehen, und der Amiga kann auch ohne Übertreibung als Vertreter einer neuen technologischen Generation bezeichnet werden. Aber an den Grundprinzipien, wie ein Computer funktioniert, hat sich trotzdem seit den ersten elektrischen Rechenmaschinen nichts geändert: Im Innersten seines Herzens, den Chips, kann jeder Computer eigentlich nur zwei Zustände voneinander unterscheiden: Nämlich "Strom fließt" und "Strom fließt nicht". Das hört sich vielleicht nach einer Binsenweisheit an, ("Ist ja klar: Kein Strom - Amiga aus. Strom vorhanden - Amiga an.") aber so meinen wir das nicht.

Aus den beiden Zuständen "an" und "aus", "Strom" und "kein Strom" leiten die elektronischen Bauteile nämlich alles andere ab. "An" und "aus" entspricht zunächst mal 0 und 1. Diese kleinste Einheit der Informationsspeicherung nennt man ein *Bit*. Durch

verschiedene Tricks und das Prinzip der *Binärzahlen* schafft es ein Computer, mehrere Bits gleichzeitig zu verarbeiten und sich so zu einem etwas höheren geistigen Horizont aufzuschwingen: Er kann mit diesen Hilfsmitteln nicht mehr nur noch 0 und 1 bzw. Strom oder kein Strom unterscheiden, sondern sogar größere Zahlen darstellen. Aber um den Schritt von den Zahlen zu Buchstaben und Zeichen zu schaffen, war noch einmal eine ganz besondere Idee notwendig: Dazu wird nämlich jedem darstellbaren Zeichen eine Zahl zwischen 0 und 255 zugeordnet. Daß es ausgerechnet 255 Zeichen sein müssen, ist ein Überbleibsel aus der Zeit, als 255 die höchste Zahl war, die ein Computer ohne besondere Anstrengungen verwalten konnte. (Für den 68000-Prozessor im Amiga ist z.B. 4294967294 die höchste Zahl, die er auf einen Schlag verstehen kann.)

Die Tabelle, nach der die Zuordnung von Zahlen und Zeichen erfolgt, ist glücklicherweise schon vor langer Zeit vereinheitlicht worden. Das ist auch gut so, sonst könnten verschiedene Computer nämlich untereinander keine Daten austauschen, wenn man sie z.B. durch ein Kabel verbindet. Und auf diese Weise kommen doch manchmal recht interessante Dinge zustande... Der Name für die Zuordnungstabelle heißt "ASCII-Code" (engl.: American Standard Code for Information Interchange, deutsch: Amerikanischer Standard-Code für Informationsaustausch). Er wurde, wie der Name schon sagt, in Amerika entwickelt und für deutsche Verhältnisse modifiziert. Eine Tabelle der ASCII-Codes finden Sie im BASIC-Referenzteil dieses Buchs beim CHR$-Befehl.

Damit wären wir wieder beim Ausgangspunkt: Der Befehl CHR$ dient dazu, aus einem ASCII-Code das zugehörige Zeichen zu machen. Und jetzt schreiten wir zu einem kleinen Experiment, mit dem wir das Innere des Rechners nach außen kehren wollen: Geben Sie bitte die folgende Zeile im Direktmodus ein. (Klicken Sie aber vorher das LIST-Window in den Hintergrund, damit Sie genug Platz auf dem Bildschirm haben.)

```
width 60 : for x=0 to 255 : ?chr$(x); : next x <RETURN>
```

Wundern Sie sich bitte nicht über den Piepser, den Sie hören werden. Der ist nämlich auch im ASCII-Code enthalten. Womit das vorhin Gesagte etwas eingeschränkt werden muß: Es gibt nicht nur auf dem Bildschirm darstellbare Zeichen, sondern auch sogenannte Steuerzeichen. CHR$(7) ist so eines. Es erzeugt den Piepser. Oder CHR$(12). Dies löscht den Bildschirm.

Aber auch wenn man sich diese Zeichen einmal wegdenkt, kann der Amiga noch ganz schön viele darstellen, oder? Sie müßten auf Ihrem Bildschirm zunächst alle Satzzeichen, Ziffern, Groß- und Kleinbuchstaben sehen und in der unteren Hälfte eine ziemliche Menge landesspezifischer Sonderzeichen.

Außerdem gibt es auch mehrfach ein kleines, eckiges Kästchen. Zu diesem Zeichen greift der Amiga immer dann, wenn er nicht weiter weiß, oder etwas fachmännischer ausgedrückt, wenn er ein Zeichen mit seinem Zeichensatz nicht darstellen kann. Wenn Sie z.B. auf die <HELP>-Taste oder auf eine der Funktionstasten in der obersten Reihe des Bildschirms drücken, erscheint dieses Kästchen. So ist es zumindest in der Version 1.0 von AmigaBASIC. Vielleicht fällt den Programmierern bei Microsoft ja in Zukunft noch eine bessere Verwendung für diese Tasten ein...

So, damit sind wir wieder zurück in unserer normalen Welt, wo wir mit Wörtern und Buchstaben ganz normal umgehen. Nach alledem, was Sie jetzt wissen, werden Sie auch den CHR$-Befehl sofort verstehen. In unserem Programm wird er verwendet, um abzuprüfen, ob bestimmte Tasten gedrückt wurden. Das geht nämlich auch. Das folgende kleine Programm wartet zum Beispiel, bis die Taste <ESC> gedrückt wird:

```
Warten:
IF INKEY$<>CHR$(27) THEN Warten
```

Dieses Beispielprogramm gehört nicht zum Videotitel-Generator. Geben Sie es jetzt also bitte nicht ein. Wenn Sie sich fragen, wie man rausbekommt, welcher Taste welche Zahl entspricht - auch dazu ist eine ASCII-Tabelle im BASIC-Referenzteil da.

Jetzt kommen wir zum eigentlichen Kern unseres Programms: Den OBJECT-Befehlen. Um bewegliche Objekte (also Bobs oder Sprites) sichtbar zu machen und zu steuern, gibt es in Amiga-BASIC eine eigene Gruppe von Anweisungen: Sie beginnen alle mit dem Wort OBJECT, dann folgt ein Punkt, und dann kommt der eigentliche Befehl. Einen von dieser Sorte kennen Sie schon, nämlich den Befehl OBJECT.SHAPE. Er dient dazu, einem Objekt eine bestimmte Form zuzuweisen. Seine Befehlssyntax sieht so aus:

OBJECT.SHAPE Objekt-Nr., Definitions-String

Von Syntax spricht man bei BASIC übrigens immer dann, wenn es um die richtige Zusammenstellung von Befehlen und den dazugehörigen Werten geht. Wenn Sie zum Beispiel im BASIC-Window

```
object.shape 1 <RETURN>
```

eingeben, verstoßen Sie gegen die richtige Syntax, weil Sie den zweiten Wert, den Definitions-String, weggelassen haben. Sie können dieses Beispiel einmal ausprobieren. Dann wird Ihnen der Amiga sagen, daß Sie einen "Syntax error" gemacht haben - also einen Syntax-Fehler. Der Ausdruck "Syntax-Fehler" kommt übrigens eigentlich aus der Sprachlehre. Es ist ein Begriff, der für falschen Satzbau steht. Wenn Sie sich AmigaBASIC wie eine Fremdsprache vorstellen, dann macht das auch Sinn. Eine falsche Zusammenstellung von Worten ist für Amiga unverständlich. Deshalb sagt er dann eben "Syntax Error" und meint, daß noch irgend etwas fehlt oder sonstwie falsch ist.

Für die Werte, die zu einem Befehl gehören, benutzt man auch den Ausdruck "Parameter". Der Wert Objekt-Nr. ist zum Beispiel einer der Parameter beim OBJECT.SHAPE-Befehl. Im Videotitel-Programm ist dieser Wert immer 1, denn wir verwenden in diesem Programm nur das Objekt mit der Nummer 1. Den Definitions-String (also in unserem Fall den Bob) kennen Sie ja schon: Er wurde mit dem Object Editor erzeugt und auf Diskette abgespeichert. Von dort kann man ihn dann in jedes BASIC-Programm einlesen.

Zwei weitere OBJECT-Befehle sind OBJECT.X und OBJECT.Y. Die brauchen Sie, um ein Objekt an eine beliebige Stelle des Bildschirms zu setzen. Dabei gibt es für jedes Objekt eine horizontale Koordinate (X) und eine vertikale (Y). Die Koordinate gibt die Lage der linken oberen Ecke des Objekts an. Der sichtbare Bereich liegt für X zwischen 0 und 617, für Y zwischen 0 und 185. "Sichtbarer Bereich" deshalb, weil Sie innerhalb dieses Bereichs das Objekt auf dem Bildschirm sehen können. Sie können auch Koordinaten außerhalb des Bildschirms angeben, das ist aber eigentlich nur solange sinnvoll, wie das Objekt noch wenigstens teilweise auf dem Bildschirm zu sehen ist. Um die Koordinaten dem Amiga mitzuteilen, werden die beiden Befehle so verwendet:

```
OBJECT.X Objekt-Nr., x-Koordinate
OBJECT.Y Objekt-Nr., y-Koordinate
```

Allerdings ist auch nach der Eingabe dieser beiden Befehlen noch nichts zu sehen von dem spannungsvoll erwarteten Objekt. Dazu benötigen Sie nämlich noch einen weiteren Befehl:

```
OBJECT.ON Objekt-Nr.
```

Damit wird ein Objekt, dessen Koordinaten dem Amiga bekannt sind, eingeschaltet und somit auf dem Bildschirm sichtbar. Das Gegenstück dazu ist übrigens:

```
OBJECT.OFF Objekt-Nr.
```

Dieser Befehl läßt das Objekt wieder vom Bildschirm verschwinden.

Nun heißen diese Objekte ja "bewegliche Objekte". Deshalb besteht Anlaß zu der Vermutung, daß es auch Befehle gibt, um sie zu bewegen.

Wer suchet, der findet. Zum Beispiel OBJECT.VX und OBJECT.VY. Das V vor X und Y steht für das englische Wort "velocity", deutsch: Geschwindigkeit. Mit OBJECT.VX und

OBJECT.VY können Sie die Geschwindigkeit angeben, in der sich das Objekt auf dem Bildschirm bewegen soll. VX gibt dabei wieder die horizontale Richtung an und VY die vertikale. Die Einheit, in der die Geschwindigkeit angegeben wird, ist aber nicht etwa km/h oder etwas ähnliches. (Obwohl Sie im Amiga sicher keine Angst vor Radarkontrollen haben müssen.) Die Geschwindigkeit wird vielmehr in Bildpunkten pro Sekunde angegeben. Sie haben sicher schon einmal gehört, daß sich die Bildschirmdarstellung bei Amiga aus einzelnen Punkten aufbaut. Über dieses Prinzip werden wir im Kapitel 2.2 noch genauer sprechen.

In der Auflösungsstufe, in der AmigaBASIC normalerweise arbeitet, stellt der Amiga 640 * 200 Punkte dar. (Berücksichtigt man den PAL-Bereich der Workbench, sind es sogar 640*256 Punkte, aber dazu kommen wir später.) Wenn man davon die Punkte abzieht, die durch den Window-Rahmen verlorengehen, kommt man genau auf den sichtbaren Bereich der OBJECT.X und OBJECT.Y-Befehle. Tatsächlich wird auch die Startkoordinate in Bildschirmpunkten angegeben.

Eine x-Geschwindigkeit von 1 Bildpunkt pro Sekunde bedeutet, daß sich das angegebene Objekt innerhalb einer Sekunde auf dem Bildschirm genau einen Bildpunkt nach rechts bewegt. Eine y-Geschwindigkeit von 2 Bildpunkten pro Sekunde bewirkt, daß sich das Objekt innerhalb einer Sekunde zwei Bildpunkte nach unten bewegt. Geben Sie gleichzeitig einen x- und einen y-Anteil der Geschwindigkeit an, bewegt sich das Objekt diagonal von links oben nach rechts unten.

"Und was wenn ich einmal ein Objekt von rechts nach links oder von unten nach oben fliegen lassen will?" Ganz einfach - dann geben Sie eine negative Geschwindigkeit an. Eine x-Geschwindigkeit von -1 Bildpunkt pro Sekunde ist sozusagen der Rückwärtsgang. Wenn Sie beim eigenen Programmieren später einmal die Vorzeichen für die richtige Richtung heraustüfteln müssen, hilft Ihnen vielleicht die folgende Skizze:

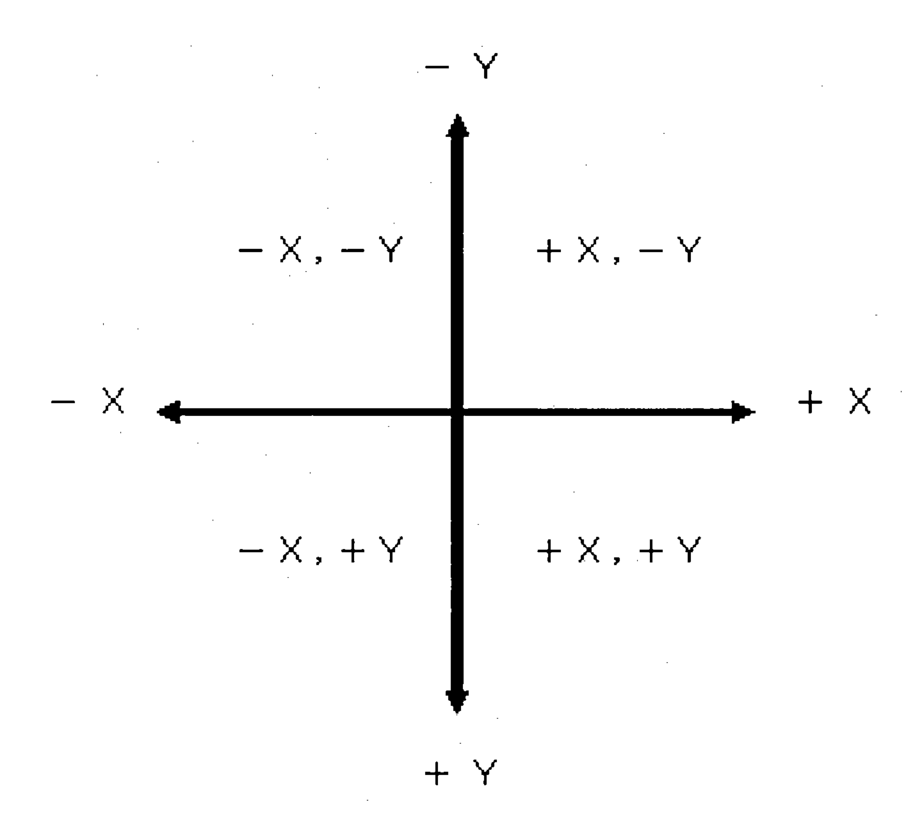


Bild 4: Die Vorzeichen für OBJECT-Geschwindigkeiten

Soll Ihr Objekt von rechts oben nach links unten fliegen, dann muß die x-Geschwindigkeit auf jeden Fall negativ sein und die y-Geschwindigkeit positiv. Die Größe der Werte hängt davon ab, wie schnell die Bewegung sein soll.

Noch ein Tip: Wenn die x-Geschwindigkeit betragsmäßig überwiegt, wird die Bewegung flacher. Ist der Betrag von y größer als der von x, bewegt sich das Objekt steiler.

Um beim Amiga aufs Gaspedal zu drücken, werden die beiden Befehle folgendermaßen verwendet:

OBJECT.VX Objekt-Nr., x-Geschwindigkeit
OBJECT.VY Objekt-Nr., y-Geschwindigkeit

Damit haben Sie aber - ähnlich wie bei OBJECT.X und OBJECT.Y - nur die Werte voreingestellt. Um das Objekt anzuschubsen, brauchen Sie noch den Befehl

OBJECT.START Objekt-Nr.

Wenn AmigaBASIC diesen Befehl findet, geht's los mit der Bewegung. Und jetzt kommt das tollste: Ab diesem Augenblick brauchen Sie sich um das Objekt gar nicht mehr zu kümmern. Sie können also Ihr BASIC-Programm irgendetwas anderes tun lassen, der Amiga sorgt selbständig für die richtige Ausführung der programmierten Bewegung. Allerdings sollte es immer eine Notbremse geben. Das heißt, eigentlich ist der nun folgende OBJECT.STOP-Befehl nicht nur eine Notbremse, sondern die Bremse schlechthin. Beim Befehl

OBJECT.STOP Objekt-Nr.

beendet das angegebene Objekt jede Bewegung und bleibt an seinem augenblicklichen Standort stehen. Zwischendurch bemerkt: Jetzt verstehen Sie sicher auch, warum man vorher beim Einlesen dem Objekt eine eigene Objektnummer zuweisen muß. Nur so ist es möglich, bestimmte Befehle auf einzelne Objekte zu beschränken.

Es gibt noch weitere Befehle zur Objekt-Steuerung. Einige davon brauchen wir in unserem Videotitel-Programm nicht. Deshalb wollen wir Sie auch an dieser Stelle nicht weiter besprechen. Sie finden jedoch eine genaue Beschreibung dieser Befehle mit Programmbeispielen im BASIC-Referenzteil unseres Buches.

Doch einen Befehl hätten wir fast vergessen: Er kommt im Programmteil 'Bewegen:' vor und heißt OBJECT.HIT. AmigaBASIC bietet die Möglichkeit, abzufragen, ob Objekte miteinander oder mit dem Bildschirmrand zusammenstoßen. Kollisions-Kontrolle

nennt man das. Die Kollision von Objekten ist eines der Ereignisse (engl.: events), die Einfluß auf ein Programm nehmen können. Zu diesem Thema können Sie beim Befehl ON COLLISION GOSUB im BASIC-Referenzteil Näheres erfahren. Allerdings müssen wir den Befehl schon jetzt einmal erwähnen, denn ohne die Angabe von OBJECT.HIT würde unser Objekt sofort anhalten, sobald es mit irgendetwas zusammenstößt. Das ist zwar ganz im Sinne der Straßenverkehrsordnung, aber in unserem Programm unerwünscht. Aus diesem Grund wird mit

```
OBJECT.HIT Objekt.Nr, 0, 0
```

festgelegt, daß unser Objekt mit allen denkbaren Gegnern bedenkenlos zusammenstoßen darf (also sowohl mit anderen Grafikobjekten, die aber in unserem Fall eh' nicht da sind, als auch mit dem Bildschirmrand).

Nachdem auch dies klar ist, stellt sich eigentlich nur noch eine kleine, fast unscheinbare Frage: Wie funktioniert das Ganze eigentlich? Was steckt hinter dem neu eingegebenen Teil des Videotitel-Programms? Bereits vorher haben wir ja abgeprüft, ob überhaupt schon ein Objekt eingelesen wurde. Falls eines vorhanden ist, werden zunächst die Variablen 'ox' und 'oy' auf den Wert 100 gesetzt. Diese Variablen helfen uns, im Auge zu behalten, an welcher Koordinate sich das Objekt befindet. Den Befehl OBJECT.HIT 1,0,0 haben wir gerade eben schon erklärt. Weiterhin sorgt der Programmteil 'Bewegen:' dafür, daß Objekt Nr. 1 eingeschaltet wird und eine eventuell noch stattfindende Bewegung aufhört. Letzteres deshalb, weil in der späteren Anwendung des Videotitel-Programms von vorherigen Aktionen noch eine Bewegung im Speicher stehen kann, die beim Einschalten des Objekts fortgesetzt würde.

Nun kann der Anwender das Grafikobjekt mit den Cursortasten steuern. Je nach dem, welche Taste gedrückt wird, ändern sich die Werte für die x- oder die y-Koordinate. Das wird im Programmteil 'Schleife:' solange wiederholt, bis die <RETURN>-Taste gedrückt wird. Dann nämlich wird die aktuelle Position als Anfangspunkt der Bewegung übernommen. Das Programm ver-

zweigt zum Teil 'Zieldef:'. An dieser Stelle tritt das Datenfeld 'Beweg' in Aktion, das wir ganz oben im Programm definiert haben. Hier werden der Reihe nach die x- und die y-Koordinaten der Zwischenpunkte unserer Bewegung abgelegt. Die Variable 'Ziel' gibt dabei Auskunft darüber, wieviele Zwischenpunkte es schon gibt. Ihr Wert wird in der ersten Position des Feldes, nämlich Beweg(0), festgehalten, weil wir diese Zahl später wieder brauchen. Die Anzahl haben wir auf insgesamt 7 Punkte beschränkt.

Die Festlegung der folgenden Zielpunkte funktioniert genauso wie die Definition des Anfangspunktes. Also mit den Cursortasten und <RETURN> zur Übernahme des festgelegten Punktes. Durch Drücken der Taste <ESC> oder nach der Eingabe des sechsten Zielpunktes wird die Eingabe beendet. Der Programmteil 'Enddef:' bringt noch mal die Anzahl der Zielpunkte auf den neuesten Stand, nimmt das Objekt vom Bildschirm und kehrt zurück zur Hauptauswahl.

Wenn Sie jetzt das Programm bis zu diesem Punkt einmal ausprobieren möchten, starten Sie es bitte. Lesen Sie dann mit der Programmoption 2 ("Object einlesen") Ihren Star ein. Oder auch irgendein anderes Grafikobjekt, wenn Sie mit dem Object Editor mehrere produziert haben sollten. Mit Option 3 ("Obj.Bewegung festlegen") können Sie dann sehen, daß Kunst frei ist: Ihr kleines Bildnis läßt sich über den Bildschirm bewegen, wie Sie wollen. Spielen Sie ruhig ein wenig damit herum, denn nur so finden Sie heraus, ob Sie bei der Eingabe vielleicht einen Fehler gemacht haben, der jetzt eine entsprechende Meldung produzieren würde. Je komplexer ein Programm wird, umso wichtiger ist dieses Herumprobieren. Um Ihnen einen etwas besseren Ausdruck dafür zu geben: Austesten. Nur für den Fall, daß jemand aus der Familie fragt...

1.16 Jetzt kommt Farbe ins Spiel - die Farbsteuerung

Der Amiga kann 4096 verschiedene Farben darstellen. Diese Nachricht war es, die wohl für die meiste Furore sorgte, als gerüchteweise die ersten Merkmale unseres Computers bekannt wurden. Mittlerweile ist das für Sie sicher keine großartige Neuigkeit mehr. Wie ist diese Farbenpracht eigentlich möglich, wenn doch mancher Computer noch vor kurzem schon ernstliche Schwierigkeiten hatte, nur 16 einigermaßen voneinander unterscheidbare Farben zu produzieren?

Einer der ausschlaggebenden Punkte ist der Monitor des Amiga. Es handelt sich dabei nämlich um einen sogenannten analogen RGB-Monitor. Sie werden jetzt zu Recht fragen, was "analog" bedeuten soll. Und "RGB"...!? (Spricht man übrigens: Er Ge Be.) Das Funktionsprinzip eines Fernsehers kennen Sie vielleicht: Ein Elektronenstrahl schreibt Zeile für Zeile des Bildes auf den Bildschirm. Durch das Auftreffen des Strahls wird eine Leuchtschicht auf dem Bildschirm zum Leuchten angeregt - die Bildpunkte entstehen auf dem Schirm. Durch die hohe Geschwindigkeit dieses Vorgangs merkt unser relativ träges Auge nicht, wie das Bild punktweise bzw. zeilenweise aufgebaut wird. Es sieht das komplette Bild.

So weit so gut. Wenn die Technik heute auch noch so simpel wäre, gäbe es nur einfarbige Monitore. Zum Beispiel die grünen Monochrom-Monitore (die toller klingen als sie sind: "Monochrom" ist griechisch und heißt "einfarbig"), die eine bekannte Computerfirma standardmäßig ihren Geräten beipackt (die sie seit einigen Jahren erstaunlich erfolgreich als "Weltstandard" verkauft).

Schon ein Schwarz/Weiß-Fernseher ist technisch anspruchsvoller. Bei ihm ist der Elektronenstrahl in der Intensität variabel, so daß beliebige Abstufungen zwischen schwarz und weiß, also Grauwerte, erreicht werden können. Wie diese Darstellungsart in Grün aussieht, können Sie sich mal ansehen, wenn Sie die Klappe an Ihrem Amiga-Monitor öffnen und die äußerst rechte Taste drücken, unter der das Wort GREEN steht. Nochmaliges Drücken dieser Taste bringt Ihnen die vertraute Farbe zurück.

Manche Leute empfinden bei langem, konzentriertem Arbeiten die Gründarstellung als angenehmer, deshalb die Umschalttaste.

Als einige geniale Leute den Farbfernseher erfanden, machten sie folgendes: Sie installierten drei Elektronenstrahlen nebeneinander, die jeweils verschieden beschaffene Punkte auf dem Bildschirm zum Leuchten brachten. Eine Art dieser Punkte leuchtet grün, eine andere rot und eine dritte blau. Rot-Grün-Blau, RGB, kommt Ihnen ein Verdacht?

Andere pfiffige Leute haben nämlich entdeckt, daß sich aus diesen drei Farben alle anderen Farben erzeugen lassen. Zumindest funktioniert das bei Fernsehern so. In einem Malkasten geht's ja etwas anders. Die drei Strahlen sind wieder in ihrer Intensität variabel. So können beliebige Farbschattierungen erzeugt werden. Diese Idee war bei Fernsehern sehr schnell erfolgreich, aber sie brauchte wieder einige Zeit, um sich bei den Computerherstellern durchzusetzen. Die Leute, deren Geistes Kind die einfarbig grüne Darstellung war, bauten jetzt nämlich Farbmonitore mit drei unveränderbar starken Stahlen für Rot, Grün und Blau. So etwas geht wegen der An/Aus-Mentalität von Computern auch recht einfach: Zwischen Computer und Monitor gibt es dann drei Leitungen für R, G und B. Das Resultat war eine Beschränkung auf acht Farben, nämlich Schwarz, Weiß, Rot, Grün, Blau, Gelb, Türkis und Violett. Diese Technik nennt man digitale Farbsteuerung. (Der Ausdruck "Digital" wird in der Computertechnik im Sinne von 0 oder 1 gebraucht. Strom oder nicht Strom, das ist hier die Frage.)

Was macht der Amiga jetzt so ganz anders? Er verwendet auch nur drei Leitungen für seine Farben. Aber jede dieser Leitungen kann eine in 16 Stufen variable Farbintensität übermitteln. Das Ergebnis liegt sehr nah an den Darstellungsmöglichkeiten eines Farbfernsehers: Ungeheure Farbenvielfalt.

Falls Sie den Amiga auch nur einige Tage besitzen, ist Ihnen das RGB-Prinzip sicher nicht mehr ganz unbekannt. Fast alle Programme bieten eine Möglichkeit zur Einstellung der Farben mit drei Reglern für den Rot-, Grün- und Blau-Anteil der Farbe. Da verhalten sich die Malprogramme Deluxe Paint oder

Graphicraft genauso wie das Voreinstellungsprogramm Preferences. Wenn Sie nach dieser ganzen Farb-Theorie jetzt ungeheuer Lust darauf verspüren, Ihre 4096 Farben auszuprobieren, können Sie ja die beiden BASIC-Windows verkleinern und auf der Workbench das Programm Preferences anklicken. Vor solchen Aktionen sollten Sie aber unbedingt die aktuelle Version des Videotitel-Programms abspeichern! Am unteren Bildschirmrand von Preferences finden Sie die drei Schieberegler. Wenn Sie genug ausprobiert haben, verlassen Sie Preferences einfach durch Klicken ins Feld "Use". Sie sollten allerdings wissen, daß damit bis zum Ausschalten des Amiga die von Ihnen getroffene Farbeinstellung als Standardeinstellung gilt.

Na, wie hat es Ihnen gefallen? Vielleicht suchen Sie jetzt ganz wild nach einem BASIC-Fenster, in dem drei Schieberegler sind. Aber das werden Sie nicht finden. Um in BASIC an diese Farbenpracht heranzukommen, muß man schon ein wenig programmieren. Aber keine Sorge: Es geht ganz einfach. Der Befehl, den Sie dazu brauchen, heißt PALETTE. Mit ihm können Sie in BASIC Farben ändern. PALETTE braucht vier Parameter. Die erste Zahl ist die Farbe, die Sie ändern wollen. Im Normalfall stehen Ihnen in AmigaBASIC vier Farben zur Verfügung. Das sind (mit den Nummern, die Sie zum Ändern angeben müssen):

Farbe	Farbnummer
blau	0
weiß	1
schwarz	2
orange	3

Die nächsten drei Zahlen geben den Prozentanteil an Rot, Grün und Blau an, der der jeweiligen Farbe zugedacht werden soll. Dieser Wert muß zwischen 0 (0% = kein Anteil) und 1 liegen (100% = voller Anteil). Probieren Sie's im BASIC-Window aus:

```
palette 0, 0.6, 0.2, 0.4 <RETURN>
```

Na? Jetzt sagen Sie nur, das wäre nicht schön. Gut, wenn's Ihnen nicht gefällt, versuchen Sie eben eigene Werte. Über Geschmack läßt sich ja bekanntlich streiten.

Sie können in BASIC übrigens die 0 vor dem Punkt (der eigentlich ein Komma sein sollte, aber der Amiga ist und bleibt halt Amerikaner) weglassen. Also geht's auch so:

```
palette 0, .2, .3, .6 <RETURN>
```

Und weil's so schön ist, wollen wir jetzt unserem Videotitel-Programm den Punkt "Farbauswahl" einbauen. Fügen Sie bitte zunächst im 'Abfrage:'-Teil die Zeile

```
if a$="4" then Farbdefinition
```

ein. Auch ganz am Anfang im Programmteil "Vorbereitungen:" brauchen wir eine kleine Erweiterung. Er sollte nach Ihren Korrekturen so aussehen:

```
Vorbereitungen:
Farben=2
d=15 : Maxfarbe=(2^Farben)-1
Textfa=1
DIM Text$(d),Farbfeld(d,3),Beweg(d),Geschw(d)
Fuell$=STRING$(16,"-")
Farbfeld(1,1)=15
Farbfeld(1,2)=15
Farbfeld(1,3)=15
```

Jetzt bitte ans Ende des Programms. Hier kommt nun wieder eine etwas längere Passage. Sie können ruhig während des Eintippens eine kleine Verschnaufpause einlegen. Wir haben Zeit. Aber vergessen Sie bitte das Abspeichern nicht.

```
Farbdefinition:
CLS:PRINT "Farbauswahl:"
Farben:
FOR x=0 TO Maxfarbe
COLOR -(x=0),x
```

```
LOCATE 5,(x*4) + 1
PRINT x;CHR$(32);CHR$(32)
NEXT x

Farbaenderung:
LOCATE 7,1 : COLOR Textfa,Hintgr
PRINT "Welche Farbe soll geändert werden?"
PRINT "(e= Ende)"; : BEEP
INPUT Antwort$
IF UCASE$(Antwort$)="E" THEN Farbzuweisung
Antwort$=LEFT$(Antwort$,2)
Farbnr=VAL(Antwort$)
IF Farbnr<0 OR Farbnr>Maxfarbe THEN BEEP: GOTO Farbaenderung

RGBRegler:
r=Farbfeld(Farbnr,1)
g=Farbfeld(Farbnr,2)
b=Farbfeld(Farbnr,3)
LOCATE 10,1 : PRINT "Rot:   <7>=- <8>=+ ";Fuell$
LOCATE 10,20+r : PRINT CHR$(124);
LOCATE 11,1 : PRINT "Grün:  <4>=- <5>=+ ";Fuell$
LOCATE 11,20+g : PRINT CHR$(124);
LOCATE 12,1 : PRINT "Blau:  <1>=- <2>=+ ";Fuell$
LOCATE 12,20+r : PRINT CHR$(124);
LOCATE 13,1 : PRINT "       <0>=Farbe o.k."
PALETTE Farbnr,r/16,g/16,b/16

Tasteneingabe:
Tast$=INKEY$
IF Tast$="" THEN Tasteneingabe
IF Tast$="7" THEN r=r-1
IF Tast$="8" THEN r=r+1
IF Tast$="4" THEN g=g-1
IF Tast$="5" THEN g=g+1
IF Tast$="1" THEN b=b-1
IF Tast$="2" THEN b=b+1
IF Tast$="0" THEN Farbaenderung

IF r<0 THEN r=0
IF r>15 THEN r=15
```

```
IF g<0 THEN g=0
IF g>15 THEN g=15
IF b<0 THEN b=0
IF b>15 THEN b=15

Farbfeld(Farbnr,1)=r
Farbfeld(Farbnr,2)=g
Farbfeld(Farbnr,3)=b
GOTO RGBRegler

Farbzuweisung:
a=Hintgr : a$="Hintergrund"
GOSUB Farbeingabe : Hintgr=a

a=Textfa : a$="Textvordergrund"
GOSUB Farbeingabe : Textfa=a

a=Texthi : a$="Texthintergrund"
GOSUB Farbeingabe : Texthi=a

COLOR Textfa,Hintgr
CLS : GOTO Anfang

Farbeingabe:
LOCATE 14,1
PRINT a$": ";a
Schleife 3:
LOCATE 14,1
PRINT a$; : INPUT Antwort$
Antwort=VAl(Antwort$)
IF Antwort$="" THEN Antwort=.5
IF Antwort<0 OR Antwort>Maxfarbe THEN BEEP : GOTO Schleife3
IF Antwort<>.5 THEN a=Antwort
RETURN
```

Sollten Sie ernsthaft Schwierigkeiten haben, den Einsatz der Befehle, die wir schon besprochen haben, zu verstehen, sollten Sie zuallererst mal im Anhang B nachschlagen. Dort finden Sie Befehlsübersichten. Wenn das nicht klappt, sollten Sie nochmals

den ausführlichen Teil, der den Befehl hier behandelt, aufmerksam durchlesen. Denn langsam müssen wir darauf bauen, daß Sie Grundfunktionen selbst verstehen.

Jetzt aber wieder ein kurzer Blick auf neue Befehle: Die Funktion STRING$(16,"-") liefert einen 16 Zeichen langen String aus lauter "-". Einen solchen String brauchen wir zur Darstellung der Farbregler. Die STRING$-Funktion ist sehr praktisch, wenn es darum geht, lange Strings zu erzeugen, in denen nur ein einziges Zeichen vorkommt. Wie wäre es zum Beispiel im BASIC-Window mit folgendem:

```
width 75 : ? string$(825,"!") <RETURN>
```

Gut, wir geben's zu, das war gemein. Ihr Amiga hatte ganz schön zu tun, nicht wahr? Und überhaupt haben wir noch einen Rüffel verdient: Wenn Sie gut aufgepaßt haben, ist Ihnen vielleicht aufgefallen, daß wir bereits zum zweiten Mal den Befehl WIDTH verwenden, ohne ihn erklärt zu haben. Wird sofort nachgeholt: WIDTH dient zur Festlegung der erlaubten Zeichen pro Zeile auf dem Bildschirm. Das ist nötig, weil der Amiga im Gegensatz zu anderen Computern am rechten Bildschirmrand nicht automatisch eine neue Zeile anfängt. Er druckt vielmehr aus dem Window heraus, und die Ausgabe ward nicht mehr gesehen. Wenn Sie's nicht glauben:

```
width 255 : ? string$(825,"!") <RETURN>
```

Je nach der verwendeten Schrift (60 oder 80 Zeichen pro Zeile, einstellbar z.B. mit Preferences) ist WIDTH 62 oder WIDTH 77 der höchste Wert, wenn alle Zeichen lesbar bleiben sollen.

Als nächstes gäbe es da den COLOR-Befehl. Er wird benötigt, wenn die Schriftfarbe geändert werden soll. Hinter COLOR werden zwei Parameter angegeben. Diese Werte geben die Farbnummer für Vordergrund (erster Wert) und Hintergrund (zweiter Wert) an. Wollen Sie also, wobei wir von der Grundbelegung der Farben ausgehen, blau auf weiß schreiben, müßten Sie eingeben:

```
color 0,1 <RETURN>
```

Übrigens: Sicher wundern Sie sich, warum wir trotz Workbench 1.2 und deutscher Tastaturanpassung im Listing 'Farbaenderung:' geschrieben haben. Der Grund ist ganz einfach: In Variablen und Labels dürfen deutsche Sonderzeichen nach wie vor nicht vorkommen, sonst gibt's einen "Syntax error".

Es folgt wieder eine Stringfunktion: UCASE$(...). Steht für "Upper Case" und heißt Großschrift. Diese Funktion setzt einen String vollständig in Großschrift um:

```
? UCASE$ ("HaeTTeN sIe's gEdAChT?") <RETURN>
```

Bleiben wir doch kurz bei den Strings. Da steht eine Zeile tiefer:

```
Farbnr=LEFT$(Antwort$,2)
```

'Antwort$' ist eine Eingabe. Möglicherweise gibt der Benutzer einen zu langen Wert ein. Das Programm interessiert sich aber nur für die ersten beiden Stellen. Um links oder rechts Teile eines Strings zu isolieren, gibt es die Befehle LEFT$ und RIGHT$. Dazu wird zum einen der String angegeben und zum anderen die Anzahl der Stellen, die abgeschnitten werden sollen. Wenn Sie's ausprobieren wollen:

```
? left$("Hallo",3), right$("Amiga",4) <RETURN>
```

"Moment mal, diesen Trick mit LEFT$ könnten wir doch im Hauptmenü unseres Programms auch gleich noch mit aufnehmen?" Stimmt, Sie haben recht. Wo wir uns doch dort so bemüht haben, alle denkbaren Fehleingaben abzufangen. Verbessern Sie bitte im 'Abfrage:'-Teil:

```
.
.
INPUT a$
a$=LEFT$(a$,1)
IF a$<"1" OR a$>"5" THEN BEEP : GOTO Abfrage
.
```

Im Programmteil 'Farbzuweisung:' findet sich ein Befehl namens GOSUB. Er sieht dem bekannten GOTO ja nicht unähnlich, aber ein kleiner Unterschied besteht doch: GOSUB ist ein Kürzel aus "go to subroutine" (deutsch: "Gehe zum Unterprogramm"). Ein Unterprogramm ist ein Programmteil, der von einem anderen Programmteil benutzt wird.

Wenn AmigaBASIC auf ein GOSUB trifft, merkt es sich die Zeile, in der es das GOSUB gefunden hat. Dann springt es in das angegebene Unterprogramm. Sobald es dort den Befehl RETURN findet, kehrt es zur ursprünglichen Zeile zurück. RETURN gehört so zu GOSUB wie NEXT zu FOR oder THEN zu IF oder Dick zu Doof: Die beiden tauchen immer zusammen auf. Wenn Sie sich jetzt immer noch nicht so ganz vorstellen können, wie das Zusammenspiel dieser beiden Anweisungen aussieht, hilft Ihnen sicher wieder der TRACE-Modus zum Verständnis.

Gleich haben wir auch diesen Teil geschafft. Es ist nur noch eine kleine Besonderheit zu erwähnen: Sehen Sie dieses "IF Antwort<>0" in der 'Schleife3:'? Was soll das heißen? Kleiner (<) und größer (>) kennen wir ja, aber beide auf einmal? "Eine Zahl kann doch gar nicht kleiner und gleichzeitig größer sein als eine andere?!" Eben. Und deshalb ist in BASIC die Kombination <> das Zeichen für "ungleich". In dem Befehl oben wird also abgeprüft, ob der Wert von 'Antwort' nicht gleich 0 ist.

Wie sieht es nun mit der Funktion unserer Farbsteuerung aus? In der Variablen 'Maxfarbe' steht, welche die höchste erlaubte Farbnummer ist. Im Moment ist das für uns die Zahl 3. Der Programmteil 'Farben:' druckt kleine Kästchen mit allen gültigen Farbnummern auf den Bildschirm. Danach wird der Anwender gefragt, welche Farbe geändert werden soll. Er kann nun die gewünschte Farbnummer oder den Buchstaben "e" eingeben. "e" bedeutet Ende, also keine weiteren Farbänderungen. Dank der

Funktion UCASE$ darf das "e" als Klein- oder Großbuchstabe eingegeben werden.

Wenn eine Farbnummer gewählt wurde, stellt der Programmteil 'RGBRegler:' ein Reglersystem für die Farbeinstellung dar. Die Regler können mit dem Ziffernblock auf der Tastatur gut bedient werden. Die Tasten <7> und <8> schieben den Rot-Regler auf und ab, <4> und <5> entsprechend den Grün-Regler und die Tasten <1> und <2> den Blau-Regler. Die Taste <0> bestätigt, daß die Farbe jetzt okay ist. In diesem Fall springt das Programm zurück und fragt, ob eine weitere Farbe verändert werden soll. Die Variablen 'r', 'g' und 'b' enthalten als Zahl zwischen 0 und 15 den jeweiligen Farbanteil und speichern ihn im Feld 'Farbfeld'.

Ein Wort zur Bedienung: Natürlich ist das Ganze nicht so komfortabel wie z.B. in Preferences. Das hat verschiedene Gründe. Ein Problem ist Ihnen vielleicht aufgefallen, falls Sie schon mit dem neuen Programmteil experimentiert haben: Wenn eine Farbe zum ersten Mal aufgerufen wird, fallen zunächst alle Regler auf 0, und die Farbe wird schwarz. Das liegt daran, daß es leider in AmigaBASIC bisher keine Möglichkeit gibt, den aktuellen Stand der Farb-Register auszulesen. Solange also das Feld 'Farbfeld' die Werte nicht kennt, sind sie für unser Programm erst mal 0. Aus diesem Grund haben wir auch in den 'Vorbereitungen:' die drei Werte fürs 'Farbfeld' stehen. Wenigstens die Schriftfarbe (weiß) soll im 'Farbfeld' vorbelegt sein, sonst würden Sie bald nur noch Schwarz sehen.

Die Tastensteuerung benötigt im Vergleich zur Farbsteuerung mit der Maus (in Programmen wie Preferences oder Graphicraft) ein wenig Übung. Bleiben Sie nicht zu lange auf den Tasten! Das Verschieben der Regler geht etwas langsam, aber solange eine Taste gedrückt bleibt, wiederholt der Amiga nach einiger Zeit (abhängig von der Preferences-Einstellung) die Tasteneingabe und merkt sie sich dann auch. Das Resultat kann sein, daß - auch wenn Sie die Tasten schon lange losgelassen haben - noch eine ganze Reihe weiterer Impulse aus dem sogenannten Tastaturpuffer kommen und die Regler sich weiterbewegen.

Am Ende der Farbeinstellung fragt Sie das Programm noch nach drei Werten: Sie können je eine Farbnummer für die Farbe des Bildschirmhintergrundes, des Textes und der Hinterlegung des Textes eingeben. Diese Farben werden dann bei der Wiedergabe des Textes verwendet, den Sie im Menüpunkt 1 eingegeben haben. Wenn Sie die Normalwerte nicht verändern wollen, drücken Sie einfach dreimal <RETURN>.

1.17 Der Mühe Lohn - das Wiedergabe-Programm

Unser Projekt "Videotitel-Programm" nähert sich deutlich seinem Abschluß. Es fehlt nur noch ein Unterprogramm, nämlich das zur Wiedergabe des gesamten bisher programmierten Titels.

Stellen Sie sich bitte einmal vor, der nächste Programmteil wäre nicht abgedruckt, und Sie müßten ihn selbst schreiben. Wie würden Sie vorgehen? Zunächst sollte man sich in so einem Fall überlegen: Was soll das Programm im einzelnen überhaupt tun? Nun, wir haben uns das so vorgestellt: Zuerst wird der Bildschirm gelöscht, und das Programm wartet darauf, daß Sie <RETURN> drücken. In dieser Zeit können Sie dann später die richtige Bandposition auf Ihrer Videocassette suchen, die Verbindung zum Amiga herstellen bzw. überprüfen und den Recorder auf Aufnahme schalten.

Nach dem Drücken von <RETURN> soll 10 Sekunden lang ein Countdown heruntergezählt werden. Das ist zum professionellen Schneiden und Überspielen von Video-Sequenzen sehr hilfreich. Danach müssen wir erst mal den programmierten Text auf den Bildschirm bringen. Und zwar in den gewählten Farben. Vielleicht schön in der Bildschirmmitte zentriert? Das sieht immer gut aus. Wenn der Text steht, soll unser Star oder ein anderes eingelesenes Objekt seine Bewegung ausführen. Während dieser Zeit bleibt der Text zum Lesen auf dem Bildschirm. Nach der Bewegung kommt dann der Abgang für unseren Star.

Wie können wir am Schluß noch mit ein wenig Effekt den Text vom Schirm entfernen? Zum Beispiel, indem wir ihn von oben und unten in die Mitte des Bildschirms rollen lassen. Das ergibt einen ähnlichen Effekt wie beim Titelbild des Programms "Amiga Tutor", das Sie vielleicht kennen. Wenn auch das geschehen ist, noch einen Augenblick andächtige Ruhe. "Aus! Gestorben! Cut. Szene im Kasten." Das war's. So also stellen wir uns ungefähr den Ablauf des letzten Programmteils vor.

Der nächste Schritt bei der Programmentwicklung ist dann eine Art Bestandsaufnahme. Was habe ich, was brauche ich? Bisher haben wir in unserem Programm folgende verwendbare Daten und Objekte erzeugt:

- Ein eingelesenes Grafikobjekt, das sich augenblicklich als Objekt Nummer 1 im Speicher die Zeit mit Warten vertreibt.

- Ein Datenfeld namens 'Beweg(x)', in dem bis zu sieben Zwischenpunkte der definierten Bewegung stehen. Die genaue Anzahl findet sich in dem Feldelement 'Beweg(0)'. Dann folgen paarweise die Koordinaten der einzelnen Punkte. Das Feld ist also so aufgebaut: (Anzahl), x1, y1, x2, y2,...

- Ein Feld namens 'Geschw(d)'. Hier sollen die x- und die y-Geschwindigkeiten abgelegt werden, mit denen sich das Objekt von einem Punkt zum jeweils nächsten bewegt. Der Aufbau dieses Feldes entspricht dem von 'Beweg(x)'. Zur Zeit ist es aber noch leer.

- Ein Datenfeld namens 'Text$', in dem bis zu 15 Zeilen Text stehen, die auf den Bildschirm gedruckt werden sollen.

- Die Farben für Text und Bildschirmhintergrund sind in den Variablen 'Hintgr', 'Textfa' und 'Texthi' abgelegt.

- Außerdem haben wir noch ein Feld namens 'Farbfeld'. Das brauchen wir jetzt aber nicht mehr, denn die Farben sind ja bereits vom System (soll heißen vom Amiga; Programmierer schieben alles, womit Sie nichts zu tun haben wollen, aufs "System") übernommen worden.

- Es gibt natürlich noch mehr Daten und Variablen, aber auch die sind in diesem Programmteil uninteressant.

Für den letzten Teil brauchen wir wieder einige neue Befehle. Die wollen wir jetzt vorstellen. Bitte versuchen Sie bei jedem neuen Befehl, einmal selbst zu überlegen, in welcher Weise wir bei der zu lösenden Aufgabe diese Befehle brauchen könnten.

Den ersten neuen Befehl brauchen wir wahrscheinlich in der Countdown-Routine. Wie soll der Amiga herausfinden, wann 10 Sekunden vorbei sind? Ganz einfach: Es gibt einige Zeit- und Datumsfunktionen in AmigaBASIC. Geben Sie den nächsten Befehl im BASIC-Window ein:

```
? date$ <RETURN>
```

Sie sehen, der String DATE$ enthält das Datum, das der Amiga gespeichert hat. Dabei zeigt er in amerikanischer Schreibweise zuerst den Monat, dann den Tag und dann das Jahr. Schön und gut, aber der Amiga soll ja keine Tage oder Monate lang warten, sondern nur Sekunden. Also probieren Sie den Nächsten:

```
? time$ <RETURN>
```

Na, das ist doch schon was. Mit diesem Befehl zeigt AmigaBASIC Stunden, Minuten und Sekunden an. Auch die Uhrzeit richtet sich nach den Einstellungen in Preferences, die Uhr wird also wahrscheinlich auch falsch gehen. Durch die Formel (VAL(RIGHT$(TIME$,2))) könnten Sie die Sekunden isolieren. Aber dieser Formelapparat sieht etwas unschön aus. Es geht auch einfacher:

```
? timer <RETURN>
```

Die Variable TIMER liefert die Zeit in Sekunden und Zehntel-Sekunden. Sie gibt die Anzahl an Sekunden an, die seit 00:00:00 (also Mitternacht) vergangen sind. Wobei allerdings Amigas Vorstellung von Mitternacht wieder einzig und allein von den Einstellungen in Preferences abhängt.

Die Rundungsfunktion werden wir bei der Berechnung der Geschwindigkeit unseres Objekts benötigen. Und weil Sie sich gerade ein bißchen daran gewöhnt haben, kommt hier noch ein Exemplar dieser mathematischen Funktionen. Aber keine Angst, es ist für die nähere Zukunft die letzte, die wir benötigen.

Was macht man, wenn man sich nur für den Betrag einer Zahl, aber nicht für das Vorzeichen interessiert? "Da war doch mal was...?" - die düstere Erinnerung an vergangene Mathematik-Stunden fördert es vielleicht an den Tag: Absolutwert nennt man so was. Gibt's in BASIC auch:

```
? abs(-3.5) : ? abs( 3.5)
```

Diese Funktion werden wir benötigen, um aus dem Abstand zweier Punkte die x- und y-Geschwindigkeiten für unser Objekt zu errechnen.

So. Weil Sie bis hierher so tapfer durchgehalten haben und alle mathematischen Funktionen über sich ergehen ließen, soll nun als keine Belohung ein neuer Grafik-Effekt vorgestellt werden. Wir haben ja vorhin schon beschrieben, wie wir den Text am Schluß wieder vom Bildschirm verschwinden lassen wollen: Der Bildschirm soll in eine obere und eine untere Hälfte aufgeteilt werden, die sich dann beide auf die Mitte zubewegen und einander sozusagen "auffressen". Für das Verschieben von Teilen des Bildschirms gibt es einen eigenen BASIC-Befehl. Geben Sie das folgende Beispiel im BASIC-Window ein:

```
for x=1 to 50 : scroll (0,0)-(630,150),2,2 : next x
```

Damit der Effekt deutlich sichtbar ist, sollte schon vorher ein wenig Text auf dem Bildschirm stehen.

Ein Teil des Bildschirminhalts sinkt langsam aber stetig nach rechts unten ab. So könnten Sie zum Beispiel den Untergang der Titanic programmieren. Den Ausdruck "scrollen" haben Sie vielleicht schon einmal gehört. Er ist ein Kunstwort, der aus dem englischen "Screen Rolling" entstand, auf Deutsch: Bildschirm-

Manchmal stört bei Berechnungen oder Vergleichen der Teil einer Zahl, der nach dem Komma steht. So auch hier, denn wir brauchen nur Sekunden und keine Zehntel-Sekunden. Die BASIC-Funktion INT(x) hilft hier weiter. Der Name kommt vom engl. "Integer" (deutsch: Ganzzahl) und ebenso wird dieser Befehl auch ausgesprochen ("Intätscher von ix"). Mit

```
? INT (23.42143)
```

wird der Nachkommateil der in Klammern angegebenen Zahl abgeschnitten. Das Ergebnis ist also 23. Aber Achtung: Durch INT wird nicht gerundet. INT(9.99999) ist 9, nicht 10. Fürs Runden kommt ein wenig später noch eine eigene Funktion.

Wenn Sie mit den jetzt vorgestellten Befehlen erreichen wollen, daß der Amiga eine Pause von genau einer Sekunde einlegt, geht das so: Eine Variable merkt sich den Wert INT(TIMER). Also die Anzahl der ganzen Sekunden seit Mitternacht. Eine Schleife muß dann den gemerkten Wert solange mit den Sekunden aus der aktuellen Zeit vergleichen, bis die beiden sich unterscheiden. Danach ist ziemlich genau eine Sekunde vergangen. In einem Listing sieht das z.B. so aus:

```
Tim=INT(TIMER)
Warten:
 IF INT(TIMER)=Tim THEN Warten
(nach einer Sekunde geht's weiter)
```

Um Sie beim Thema Runden von Zahlen nicht zu lange auf die Folter zu spannen: Die benötigte Funktion heißt CINT ("Convert to Integer", deutsch: "Wandle in eine ganze Zahl um"). Das ist zwar in Bezug auf die Namensgebung nicht übermäßig logisch, aber die Kenntnis dieser Funktion ist oft sehr hilfreich. CINT(x) rundet nach folgendem Schema: Bis einschließlich 4,5000... wird auf 4 abgerundet, größere Zahlen (also ab 4,5000...01) werden auf 5 aufgerundet. Vergessen Sie nicht, daß nach der amerikanischen Schreibweise das Komma ein Punkt ist. Im BASIC-Window können Sie vergleichen:

```
? cint (4.6) : ? int (4.6)
```

rollen. Wenn Sie zum Beispiel im BASIC-Window in der untersten Zeile eine Eingabe machen, scrollt der Bildschirm nach Drücken der <RETURN>-Taste eine Zeile nach oben. Dabei wird die erste Bildschirm-Zeile über den oberen Bildschirmrand hinausgeschoben, dafür erscheint am unteren Rand eine neue Zeile.

Das Verschieben des Bildschirminhalts beherrscht der Amiga nicht nur nach oben, sondern in alle Richtungen. Die Syntax des SCROLL-Befehls sieht so aus:

```
SCROLL (x1,y1)-(x2,y2), x-Richtung, y-Richtung
```

Mit dem Ausdruck (x1,y1)-(x2,y2) geben Sie die Begrenzungspunkte des Rechtecks an, das verschoben werden soll. (x1,y1) sind die Koordinaten des Punktes, der das Rechteck links oben begrenzt, (x2,y2) gibt die Koordinaten des Punktes an, der sich am weitesten rechts unten im Rechteck befindet. Die Werte für x-Richtung und y-Richtung legen genau wie bei den OBJECT-Befehlen fest, um wieviele Bildschirmpunkte und in welche Richtung der Ausschnitt verschoben werden soll. Ein einmaliger Aufruf von SCROLL bewirkt allerdings nur, daß das Rechteck sofort an seinem neuen Platz erscheint. Um eine gleichmäßige Bewegung zu erreichen, muß man daher, wie in unserem Beispiel, mit einer FOR-NEXT-Schleife oder einer anderen Schleifenkonstruktion dafür sorgen, daß SCROLL mehrfach ausgeführt wird. Mit Hilfe dieser Funktion werden wir am Ende der Wiedergabe den Bildschirminhalt verschwinden lassen.

Das wären eigentlich alle neuen Befehle, die wir im nächsten Teil des Programms brauchen. Wenn Sie jetzt den nächsten Teil des Videotitel-Programms eingeben, versuchen Sie bitte, bereits beim Eingeben zu verstehen, was die einzelnen Befehle und Programmabschnitte bewirken. Was wir erreichen wollen, und wie es aussehen soll, haben wir ja schon vorhin beschrieben.

Vergessen Sie aber bitte trotzdem nicht, das Programm zwischendrin immer mal wieder auf Diskette abzuspeichern.

```
Wiedergabe:
CLS
PRINT "Bitte drücken Sie <RETURN>"
PRINT "für Beginn der Wiedergabe."
Taste:
a$=INKEY$
IF a$=CHR$(13) THEN CLS : c=10 : GOTO Countdown
GOTO Taste

Countdown:
LOCATE 10,38 : PRINT c
c=c-1 : IF c<0 THEN WiedergStart
Tim=INT(TIMER)
Warten:
IF INT(TIMER)=Tim THEN Warten
GOTO Countdown

WiedergStart:
WIDTH 80
COLOR Textfa,Hintgr : CLS
COLOR Textfa,Texthi
FOR x=1 TO Zeilen
Text$=LEFT$(Text$(x),78)
h=INT((80-LEN(Text$))/2)+2
LOCATE x+17-Zeilen,h : PRINT Text$
NEXT x
COLOR Textfa,Hintgr
IF Beweg(0)=0 THEN Allesvorbei

OBJECT.X 1,Beweg(1)
OBJECT.Y 1,Beweg(2)
OBJECT.ON 1
FOR x=1 TO Beweg(0)-1
OBJECT.STOP 1
GOSUB GeschwBerechnen
OBJECT.X 1,Beweg(x*2-1)
OBJECT.Y 1,Beweg(x*2)
OBJECT.VX 1,Geschw(x*2-1)
OBJECT.VY 1,Geschw(x*2)
OBJECT.HIT 1,0,0
```

```
OBJECT.START 1

Tst=TIMER
Schleife4:
px=ABS(Beweg(x*2+1)-OBJECT.X(1))
py=ABS(Beweg(x*2+2)-OBJECT.Y(1))
IF INT(TIMER-Tst)<18 AND (px>15 OR py>15) THEN Schleife4
NEXT x
OBJECT.OFF 1

Allesvorbei:
Tst=TIMER
IF Beweg(0)<>0 THEN Abgang
Warten2:
IF TIMER-Tst < (2*Zeilen+2) THEN Warten2
Abgang:
FOR x=1 TO 30
SCROLL (1,1)-(630,100),0,3
SCROLL (1,100)-(630,180),0,-3
NEXT x
COLOR Textfa,Hintgr
CLS : GOTO Anfang

GeschwBerechnen:
ox=OBJECT.X (1) : oy=OBJECT.Y (1)
Beweg(x*2-1)=ox : Beweg(x*2)=oy
zx=Beweg(x*2+1) : zy=Beweg(x*2+2)
FOR xx=1 TO 64 STEP .2
Geschw(x*2-1)=CINT((zx-ox)/xx)
Geschw(x*2)=CINT((zy-oy)/xx)
IF ABS(Geschw(x*2-1))<40 AND ABS(Geschw(x*2))<40 THEN xx=64
NEXT xx
RETURN
```

Wenn Sie alles haben, bitte gleich abspeichern. Aber das kennen Sie ja mittlerweile.

Haben Sie im Groben verstanden, was das Programm tut? Wir wollen jetzt auf jeden Fall noch mal die einzelnen Abschnitte besprechen. Vergleichen Sie die Erklärungen mit dem, was Sie

sich schon selbst zusammengereimt haben - aber nicht schummeln!

Der Programmteil 'Wiedergabe:' wartet, bis die <RETURN>-Taste gedrückt wird. CHR$(13) entspricht dieser Taste. Dann wird die Variable 'c' auf 10 gesetzt. Sie ist die Zählvariable für unseren Countdown. Deshalb wird sie im Teil 'Countdown:' auch in die Mitte des Bildschirms gedruckt. Den Wert 'c' vermindern wir nach einer Sekunde immer um 1, bis der Wert 0 erreicht ist. Die Schleife 'Warten:' sorgt für eine Sekunde Verzögerung, bis der nächste Wert von 'c' gedruckt wird.

'WiedergStart:' setzt zunächst die Zeilenbreite auf 80 Zeichen, löscht dann den Bildschirm in der definierten Hintergrundfarbe und stellt die Farbe zur Wiedergabe der Zeichen ein, wie sie im Farbzuweisungs-Teil festgelegt wurde.

Die Schleife, die dann kommt, zentriert den Text jeweils innerhalb einer Zeile. Dazu wird der Text mit LEFT$ zunächst auf 78 Zeichen begrenzt. Längere Eingaben werden abgeschnitten. Merken Sie sich daher bitte gleich, daß Sie im Texteingabe-Teil nicht mehr als 78 Zeichen pro Zeile eingeben sollten.

Die Variable 'h' errechnet dann die Position innerhalb der Zeile, ab der der Text beginnen soll. Dazu wird der verbleibende Platz (80-LEN(Text$)) einfach halbiert - eine Hälfte für links vom Text und eine für rechts. Schließlich wird der Text an die so errechnete Position gedruckt und danach die Farbe wieder zurückgesetzt. Wurde keine Bewegung definiert (zu überprüfen durch die Bedingung Beweg(0)=0), springt das Programm gleich zum Teil 'Allesvorbei:'.Andernfalls wird das Objekt auf seine Anfangsposition gebracht und die Bewegungsschleife beginnt.

Bei jedem Schleifendurchlauf wird das Objekt zunächst angehalten, und seine Geschwindigkeitswerte werden berechnet (dazu gleich mehr). Die errechneten Werte werden übergeben, die Kollisionsabfrage wird mit OBJECT.HIT 1,0,0 ausgeschaltet und die definierte Bewegung gestartet. Die 'Schleife4:' läuft so lange, bis das Objekt auf 15 Bildpunkte an seine nächste Zielposition herangekommen ist. Dieser Wert hat sich bei Experimenten als

sinnvoll erwiesen. Es kann jedoch vorkommen, daß das Objekt wegen der Rundung bei der Geschwindigkeitsberechnung sein Ziel verpaßt. Deshalb läuft noch eine Zeitkontrolle mit: Wenn das Objekt länger als 18 Sekunden von einem Punkt zum nächsten braucht, wird die Bewegung abgebrochen und die nachfolgende eingeleitet. 18 Sekunden sind die Zeit, die das Objekt für die weitestmögliche Entfernung innerhalb des Bildschirms braucht, nämlich von einem Eck zum diagonal gegenüber-liegenden.

Wie funktioniert das Unterprogramm 'GeschwBerechnen:'? Es stellt zunächst die aktuellen Koordinaten fest, die ja nicht hundertprozentig mit den im Feld 'Beweg(x)' gespeicherten Werten übereinstimmen. Deshalb werden die tatsächlichen Werte anstelle der "theoretischen" ins Feld übernommen. Sie sind gleichzeitig die Anfangskoordinaten der nächsten Bewegung und werden deshalb den Variablen 'ox' und 'oy' übergeben. Die Endkoordinaten stehen an den nächsten beiden Positionen im Feld 'Beweg(x)'.

Die folgende FOR...NEXT-Schleife dividiert die Entfernung zwischen den x- und den y-Koordinaten solange durch immer höhere Zahlen, bis beide Werte kleiner als 40 sind. Sonst wird die Bewegung nämlich zu schnell. Das Dividieren durch die gleiche Zahl bewirkt, daß das Verhältnis zwischen der x- und der y-Entfernung konstant bleibt. Dieses Verhältnis muß bei der Geschwindigkeit übernommen werden, damit die Bewegungsrichtung stimmt. Sobald die Geschwindigkeit in der richtigen Größenordnung liegt, wird 'xx' auf 64 gesetzt, wodurch die Schleife beendet wird. (FOR xx=1 TO 64 STEP .2 erhöht xx solange um 0.2, bis xx den Wert 64 annimmt. Sobald xx den Wert 64 hat, ist Schluß.)

Zurück zum Hauptprogramm (RETURN heißt das auf BASIC). Der Abschnitt 'Allesvorbei:' fehlt uns noch. Falls keine Objektbewegung stattfand, hatte der Zuschauer ja keine Zeit, den Text zu lesen. Deshalb wartet das Programm, und zwar pro Zeile Text 2 Sekunden plus 2 Extra-Sekunden zur allgemeinen Besinnung.

Am Ende folgt unsere SCROLL-Routine, die die beiden Bildhälften mit entgegengesetzten x-Richtungen aufeinander zu bewegt. So verschwinden dann Text und Hintergund vom Schirm. Noch schnell Farben zurücksetzen und Bildschirm löschen, falls noch irgendwo ein Stückchen Text oder Hintergrund übrig sein sollte, und dann schleunigst zurück zur Hauptauswahl.

Tja, so funktioniert's. Oh, fast hätten wir etwas Wichtiges vergessen: Der neue Programmteil muß ja auch noch in die Auswahl aufgenommen werden. Fügen Sie bitte im Programmteil 'Abfrage:' unter der Zeile

```
IF a$="4" THEN Farbdefinition
```

die Zeile

```
IF a$="5" THEN Wiedergabe
```

ein. Jetzt brennen Sie natürlich darauf, das Programm auszuprobieren. Nur zu! Geben Sie zuerst mit der Option 1 den Text ein, der gezeigt werden soll. Kleiner Tip: Einige Leerzeilen erhöhen die Lesbarkeit.

Mit Option 2 lesen Sie dann das Objekt von Diskette ein, das Sie im Vordergrund bewegen wollen. Um diese Bewegung festzulegen, wählen Sie Menüpunkt 3.

Falls Ihnen die Farben nicht zusagen, bitte Option 4 verwenden. Denken Sie daran, daß Sie nach Eingabe von "e" noch die Textfarben ändern können. Die erste eingegebene Farbnummer legt den Bildschirmhintergrund fest, die zweite die eigentliche Textfarbe. Mit der dritten können Sie den Text noch mit einem farbigen Balken unterlegen. Wenn Sie nur <RETURN> drücken, werden die angezeigten voreingestellten Werte übernommen. Option 5 schließlich spielt den ganzen Titel ab. Für dieses Programm gilt ganz besonders: Probieren geht über studieren. Was Ihnen nicht gefällt, können Sie über die Menüpunkte ändern.

Wenn Sie die letzten Eingabefehler verbessert haben, speichern Sie bitte eine endgültige Version des Programms auf Diskette ab.

Wir werden noch an verschiedenen Stellen dieses Buches Änderungen am Videotitel-Programm durchführen, wenn wir neue Möglichkeiten kennengelernt haben, die das Programm komfortabler, schöner oder interessanter machen. Zum Beispiel wenn wir als Hintergrund Bilder aus... Nein. Das wird jetzt noch nicht verraten. Fürs erste ist es fertig. Bis zum nächsten Kapitel wünschen wir Ihnen viel Spaß beim Experimentieren.

2. Mehr Farbe auf den Schirm! - Farben und Auflösungsstufen

Vielleicht sagen Sie sich: "Da habe ich einen Computer, der 32 Farben gleichzeitig darstellen kann, und die sogar aus 4096 möglichen. Und was habe ich bisher davon gesehen? Ganze vier Stück."

Die vier Farben, die wir bisher verwendeten, können Sie zwar mit dem PALETTE-Befehl ändern. Das Videotitelprogramm bietet diese Möglichkeit zum Beispiel mit der Option 2 ("Farben festlegen"). Aber mehr als vier Farben gleichzeitig haben wir, zumindest von AmigaBASIC aus, bisher noch nicht geschafft. Das soll sich ändern. Die nächsten Kapitel stellen Ihnen die ganze Farbenpracht Ihres Computers vor. Bleibt uns nur, zu hoffen, daß Ihnen das Ganze jetzt nicht zu bunt wird...

2.1 Die zweite Kostprobe: Farbige Zeiten

Sie wissen ja noch: Unsere Kostproben wollen Ihnen zum Beginn eines Kapitels zeigen, daß manche Effekte in AmigaBASIC mit sehr geringem Aufwand zu erreichen sind. Sie brauchen diese Beispiele noch nicht auf Anhieb zu verstehen. Tippen Sie's einfach ein und schauen Sie sich das Ergebnis an.

```
SCREEN 1,320,200,5,1
WINDOW 2,"Bunter geht's nicht.",,23,1
FOR x=0 TO 7
FOR y=0 TO 3
Fa=x+y*8
LINE (x*38,y*45)-((x+1)*38,(y+1)*45),Fa,bf
NEXT y,x
WHILE INKEY$=""
PALETTE INT(31*RND)+1,RND,RND,RND
WEND
WINDOW CLOSE 2
SCREEN CLOSE 1
WINDOW 1
```

Falls Sie sich wundern: In der zweiten Zeile handelt es sich nicht um einen Druckfehler! Nach den Anführungszeichen kommen wirklich zwei Kommas, dann erst die Zahl 23. Vergessen Sie bitte nicht, das Programm nach dem Eingeben gleich auf Diskette abzuspeichern. Sie können es dann später zum Experimentieren benutzen, wenn Sie genauer verstehen, wie alles funktioniert.

Zum Ausprobieren noch einige Tips: Verändern Sie ruhig mal die Größe des Windows. Sie können es dann auch verschieben. Werfen Sie auch einen Blick auf das Aussehen der Pulldown-Menüs. Um das Programm zu beenden, drücken Sie einfach eine beliebige Taste auf der Tastatur.

2.2 Amiga löst sich auf - die Auflösungsstufen

Sie haben in der Kostprobe ein kleines Programm gesehen, das farbige Flächen auf dem Bildschirm auf Zufallsbasis verändert. AmigaBASIC bietet Befehle, mit denen es möglich ist, ein BASIC-Programm beliebig viele Windows verwalten zu lassen. Diese Windows können sich auf einem von insgesamt fünf *Screens* befinden. Wenn Ihnen dieser Ausdruck nichts sagt, gleich werden Sie mehr darüber erfahren. Für die von BASIC erzeugten Screens und Windows können verschiedene Auflösungsstufen und verschiedene Anzahlen an möglichen Farben verwendet werden.

Bevor wir Ihnen die einzelnen Befehle vorstellen, benötigen Sie aber noch einige Informationen zum Thema Farben und Auflösung. Irgendwie muß das Darstellen von Farben und Grafiken wohl mit Speicherkapazität zusammenhängen. Denn wenn Sie Besitzer eines 256 KByte-Amiga sind oder einmal waren (es wäre dann nämlich wirklich empfehlenswert, sich so schnell wie möglich zumindest die Erweiterung auf 512 KByte RAM zuzulegen), wissen Sie, daß es Grafiken und Grafikprogramme gibt, die der Amiga ohne Speichererweiterung gar nicht darstellen kann.

Das ist auch verständlich, denn ein Computer muß sich ständig merken, wie der Bildschirm gerade aussieht. Sonst könnte er Ihre Eingaben nicht darstellen und keine Grafik-Animation erzeugen. (Erinnern Sie sich an die Bobs? Die müssen auch erstmal in den Speicher gebracht werden. Dann werden sie in die aktuelle Bildschirmdarstellung hineinkopiert und vor der nächsten Phase der Bewegung wieder vom Bildschirm weggenommen.)

Ihr Amiga wüßte auch nicht, welche Zeichen oder Grafiken er noch gerade eben selbst auf den Bildschirm gebracht hat. Und weil der einzige Ort, wo sich ein Computer etwas merken kann, seine Speicherbausteine sind, zwackt halt jede Grafik und jede Farbe wieder ein Stück von diesem Speicher ab.

Die Bildschirmdarstellung eines Computers besteht aus einer Unmenge einzelner kleiner Punkte. Sie werden *Pixels* genannt. Das Bild auf dem Schirm setzt sich aus Pixels zusammen wie ein Foto in der Zeitung aus Rasterpunkten: Wenn die Punkte klein genug sind und der Betrachter eine gewisse Entfernung zu ihnen hat, verschmelzen die einzelnen Punkte zu einem Gesamtbild. Dieses Prinzip haben Sie in diesem Buch schon einmal kennengelernt, als es um die Funktionsweise eines Fernsehers ging.

Bei einfarbigen Darstellungen kann ein Pixel nur zwei Zustände haben. Entweder ist es an, also sichtbar, oder es ist aus, also unsichtbar. Deshalb lassen sich Pixels auf dem Bildschirm hervorragend durch Bits im Speicher darstellen. Im Speicher des Amiga entspricht jeder Punkt auf dem Bildschirm einem Bit.

Der Amiga bietet vier verschiedene Auflösungsstufen. In jeder wird eine andere Anzahl an Pixels auf dem Bildschirm dargestellt. Die niedrigste Auflösungsstufe beträgt 320 Pixels horizontal und 200 Pixels vertikal. In dieser Stufe lief z.B. die Kostprobe zu diesem Kapitel. Insgesamt befinden sich in diesem Modus 320 mal 200, also 64.000 Pixels auf dem Bildschirm. Da der Speicherbedarf von Daten, Programmen und Grafiken meistens in Byte angegeben wird, und ein Byte aus 8 Bit besteht, sind das 8.000 Byte. Also rund 8 KByte. (Warum nur "rund" und nicht genau, erfahren die Interessierten unter dem Stichwort *Byte* im Fachwortlexikon.)

Die nächsthöhere Stufe hat bereits 640 Pixels horizontal und ebenfalls wieder 200 Pixels vertikal. In dieser Stufe arbeitet der Amiga meistens. Die Workbench läuft in dieser Auflösung, ebenso AmigaBASIC. 640 mal 200 Pixels sind immerhin schon 128.000 Pixels, also rund 16 KByte.

Es gibt zwei noch höhere Stufen. Sie werden im *Interlace*-Modus erzeugt, den wir etwas später besprechen werden. Ihre Auflösung beträgt im ersten Fall 320 * 400 Pixels (128.000 Stück bzw. 16 KByte) und im zweiten Fall 640 * 400 Pixels (256.000 Einzelpunkte oder 32 KByte).

Alle Angaben über Auflösungsstufen, die wir Ihnen gerade erklärt haben, beziehen sich auf die Bildschirmdarstellung, wie sie für amerikanische Amiga-Progamme üblich ist. Da AmigaBASIC von einer amerikanischen Softwarefirma programmiert wurde, wurden auch die dort üblichen Standardwerte verwendet. Nun wissen Sie aber sicher, daß, seitdem es die Workbench 1.2 gibt, der sogenannte PAL-Bereich des Amiga-Bildschirms unterstützt wird. Das hat nichts mit Hundefutter zu tun, es geht vielmehr um das deutsche Farbfernsehsystem. Das PAL-Fernsehsystem, das in der Bundesrepublik und vielen anderen europäischen Ländern verwendet wird, benutzt mehr Bildschirmzeilen als das (amerikanische) NTSC-System. Deshalb blieb zu Zeiten der Workbench 1.1 auf PAL-Bildschirmen etwa das untere Siebtel ungenutzt. In Zusammenarbeit mit der Workbench-Version 1.2 kann der PAL-Amiga diesen zusätzlichen Bildschirmbereich nutzen. So ergibt sich dann im niedrigauflösenden Modus eine Auflösung von 320*256 Punkten, in der mittleren Auflösung sind es 640*256 Punkte und im Interlace-Modus 320*512 bzw. 640*512. Natürlich kostet so ein PAL-Bildschirm dann auch mehr Speicherplatz, aber dazu kommen wir etwas später.

Vielleicht fragen Sie sich jetzt, ob denn Programme, die aus Amerika stammen, mit diesem PAL-Bildschirm überhaupt funktionieren. Wir können Sie beruhigen: Der Amiga kann selbständig erkennen, ob ein Programm den PAL-Bereich nutzt oder nicht. Wenn nein, wird einfach die amerikanische Standard-Auflösung verwendet.

Nur AmigaBASIC bewegt sich ein wenig zwischen den Fronten. Das BASIC-Window kann normalerweise nicht höher als 200 Pixels sein. Sie können das Window aber so verschieben, daß ein Teil von ihm im PAL-Bereich dargestellt wird. Das LIST-Window hingegen kann, wenn Sie wollen, so vergrößert werden, daß es den gesamten PAL-Bildschirm einnimmt. Was es sonst noch zu beachten gibt, werden wir Ihnen später erzählen.

Egal ob PAL oder nicht, bei allen Angaben über Speicherplatz sind wir bisher davon ausgegangen, daß wir nur eine Farbe verwenden. Oder eigentlich zwei, wenn man's genau nimmt: Eine für den Vordergrund, also den Text und die Grafiken (z.B. weiß), und eine für den Hintergrund (z.B. blau). Wenn ein Punkt blau sein soll, ist das zugehörige Bit im Speicher aus, das heißt: nicht gesetzt. Soll der Punkt weiß sein, ist es an bzw. gesetzt. Bis dahin ist es für den Amiga problemlos möglich, mit einem Bit pro Bildpunkt auszukommen. Wie bekommt man aber jetzt vier, acht, ja sogar bis zu 32 Farben?

Der Trick, der dazu verwendet wird, ist ganz einfach: Wo ein Bit nicht ausreicht, nimmt man weitere dazu. Wenn man zwei Bits für einen Punkt zur Verfügung hat, kann man schon vier Farben unterscheiden: Sind beide Bits aus, soll zum Beispiel blau dargestellt werden. Ist das erste Bit aus, das zweite aber an, vielleicht weiß. Ist dagegen das erste Bit an, und das zweite aus, könnte Schwarz dazugenommen werden. Und sind beide Bits an, kann eine vierte Farbe, z.B. Orange, erscheinen.

Genauso wie gerade beschrieben, werden die Pixels von vierfarbigen Bildern gespeichert, wie Sie sie auf der Workbench oder im LIST- und BASIC-Window von AmigaBASIC sehen. Allerdings braucht ein solcher Bildschirm zur Speicherung dann nicht mehr nur 16 KByte, sondern das doppelte, nämlich 32 KByte.

Wenn man jetzt noch mehr Farben will, kommen eben weitere Bits dazu. Mit drei Bits können schon 8 Farben erzeugt werden. Spielen Sie mal die Kombinationen durch: aus-aus-aus , aus-aus-an, aus-an-aus, aus-an-an, an-aus-aus, an-aus-an, an-an-aus und an-an-an. Weil Ihnen danach wahrscheinlich vor lauter

auns und as' - pardon: ans und aus' - schon schwindelig ist, schreibt man für ein "aus"-geschaltetes Bit eine 0 und für ein "an"-geschaltetes eine 1. Damit sind wir dann auch schon fast bei den *Binärzahlen*, die Sie im Zwischenspiel 4 näher kennenlernen werden. Aber haben Sie bis dahin ruhig noch etwas Geduld. Die acht Möglichkeiten, die sich aus drei Bits ergeben, sehen so aus:

000 001 010 011 100 101 110 111

Das ganze Spielchen läßt sich genauso für vier und fünf Bits fortsetzen. Und theoretisch noch viel weiter. Mit nur 12 Bits könnte man alle 4096 Farben des Amiga verschlüsseln, aber da spielen die Bausteine zur Video-Erzeugung nicht mehr mit. Nach 32 Farben ist im Normalfall Schluß.

Die folgende Tabelle zeigt alle auf dem Amiga möglichen Auflösungsstufen, die erlaubten Farben und den dazu nötigen Speicher.

Auflösung	Bezeichnung	Farben	Bits/Pixel	Speicherbedarf
320 * 200	Niedrig-	2	1	8 KByte
	auflösend,	4	2	16 KByte
	Normal	8	3	24 KByte
		16	4	32 KByte
		32	5	40 KByte
640 * 200	Hoch-	2	1	16 KByte
	auflösend,	4	2	32 KByte
	Normal	8	3	48 KByte
		16	4	64 KByte
320 * 400	Niedrig-	2	1	16 KByte
	auflösend,	4	2	32 KByte
	Interlace	8	3	48 KByte
		16	4	64 KByte
		32	5	80 KByte
640 * 400	Hoch-	2	1	32 KByte
	auflösend,	4	2	64 KByte
	Interlace	8	3	96 KByte
		16	4	128 KByte

Tabelle 2: **Die Auflösungsstufen des Amiga bei Standard-Auflösung**

Auflösung	Bezeichnung	Farben	Bits/Pixel	Speicherbedarf
320 * 256	Niedrig-	2	1	10 KByte
	auflösend,	4	2	20 KByte
	Normal	8	3	30 KByte
		16	4	40 KByte
		32	5	50 KByte
640 * 256	Hoch-	2	1	20 KByte
	auflösend,	4	2	40 KByte
	Normal	8	3	60 KByte
		16	4	80 KByte
320 * 512	Niedrig-	2	1	20 KByte
	auflösend,	4	2	40 KByte
	Interlace	8	3	60 KByte
		16	4	80 KByte
		32	5	100 KByte
640 * 512	Hoch-	2	1	40 KByte
	auflösend,	4	2	80 KByte
	Interlace	8	3	120 KByte
		16	4	160 KByte

Tabelle 2a: **Die Auflösungsstufen des Amiga bei PAL-Auflösung**

Na, ist Ihnen jetzt klar, warum sich die Computerindustrie die Jagd nach immer mehr Speicherplatz für immer weniger Geld auf die Fahnen geschrieben hat? Jetzt verstehen Sie sicher auch, warum man 512 KByte Speicher problemlos mit wenigen Bildern auffüllen kann. Aber zurück zur vorerst noch etwas graueren Theorie.

Die Bits, die für ein bestimmtes Pixel zuständig sind, befinden sich nicht unmittelbar hintereinander im Speicher. Weil es für die Speicherverwaltung des Amiga günstiger ist, liegen sie in sogenannten *Bit-Ebenen.* Bei 2 möglichen Farben gibt es nur eine Bit-Ebene. Für vier Farben kommt eine weitere dazu. Das erste

Bit des jeweiligen Pixels liegt in der ersten Bit-Ebene, das zweite Bit in der zweiten. Kommt ein drittes, viertes oder fünftes Bit pro Bildpunkt dazu, wird dafür eben auch eine dritte, vierte oder fünfte Bit-Ebene angelegt.

Um die Funktion der Bit-Ebenen besser zu verstehen, kann man sie sich untereinander angeordnet vorstellen: Je mehr Ebenen untereinander liegen, umso mehr Farben können dargestellt werden.

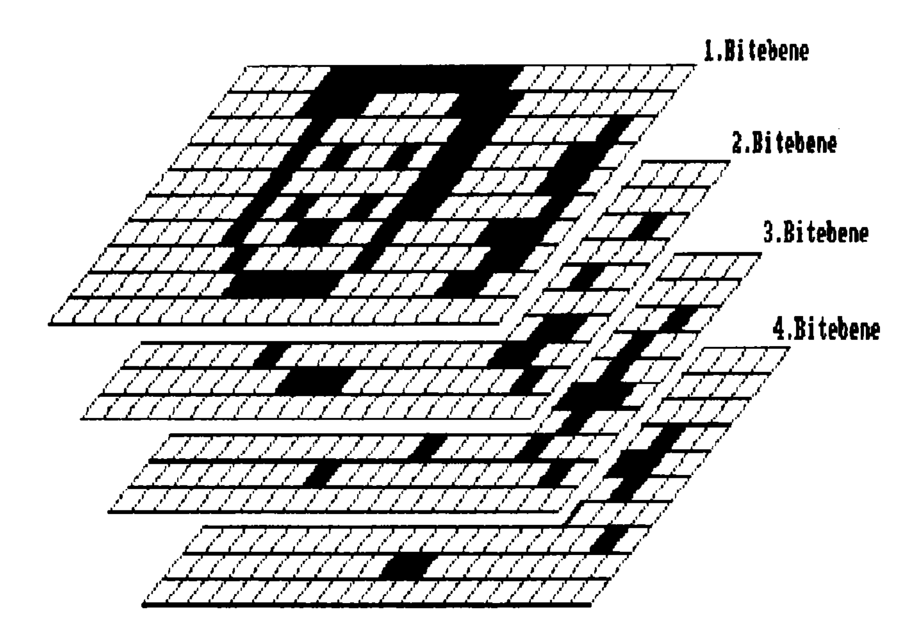


Bild 5: **Die untereinander angeordneten Bit-Ebenen speichern ein farbiges Bild**

Streng genommen stimmt dieses Bild nicht ganz, denn die Bit-Ebenen liegen im Speicher hintereinander. Zuerst also alle Bits der ersten Ebene, daran anschließend alle Bits der zweiten Ebene usw. Zur besseren Vorstellung ist das Bild aber gut geeignet.

Wenn Sie jetzt sagen, "Von alldem habe ich höchstens die Hälfte verstanden - und auch die nur so halb." macht das auch nichts: Um in BASIC Farben programmieren zu können, müssen Sie über solche Details der Speicherverwaltung nicht unbedingt Bescheid wissen. Es ist nur wichtig, daß Sie eine Vorstellung davon haben, wieviele Bit-Ebenen für wieviele Farben nötig sind, und wieviel Speicherplatz Sie das ungefähr kostet. Bei allzu verschwenderischer Benutzung von Farben und Screens sind nämlich auch 512 KByte RAM sehr schnell verbraucht.

Wir haben gerade schon wieder von "Screens" gesprochen. Was ein "Screen" genau ist, und wozu er benutzt wird, erfahren Sie im nächsten Kapitel.

2.3 Wir fallen aus dem Rahmen - Screens

Vielleicht haben Sie das schon einmal bei einem Grafik-Programm oder einer Amiga-Vorführung gesehen: Da nimmt einer die Maus und zieht auf einmal den ganzen Bildschirm nach unten. Dahinter taucht dann ein völlig anderes Bild auf, zum Beispiel der berühmte springende Ball aus den Workbench-Demos, der beim Aufprall immer so klingt wie ein Elefant beim Seilspringen.

Und als ob es noch nicht genug wäre, kommen manchmal dahinter noch weitere Grafik-Bildschirme zum Vorschein, auf denen sich wieder andere Sachen tummeln. Ganz am Schluß schließlich, wenn auch der letzte Bildschirm in der Versenkung verschwunden ist, sieht man nur noch einen blauen leeren Bildschirm und staunt ob der Dinge, die da vorher waren.

Das, was da verschoben wird, nennt man "Screen". Auf Deutsch heißt das nichts anderes als "Bildschirm". Machen Sie doch einmal das folgende Experiment mit: Verkleinern Sie das LIST- und das BASIC-Window so, daß Sie die weiße Kopfleiste der Workbench sehen können. Bewegen Sie dann den Mauszeiger auf die Leiste und ziehen Sie mit gedrückter linker Maustaste die Kopfleiste nach unten: Sie sehen, daß der ganze Bildschirminhalt mit allen Windows und Icons nach unten wandert und am unte-

ren Bildschirmrand verschwindet. Sie haben soeben den Workbench-Screen bewegt. Dahinter wird sich im Normalfall nichts mehr befinden. Falls doch, laufen vielleicht noch andere Programme auf Ihrem Amiga, oder von Ihren BASIC-Experimenten ist noch ein Screen geöffnet.

Auf einem Screen können sich beliebig viele Windows und Icons befinden, allerdings ist die Auflösungsstufe und die Anzahl der höchstmöglichen Farben für die Objekte auf einem Screen immer dieselbe. Das liegt daran, daß diese Werte beim Anlegen eines Screens angegeben werden müssen. Sie gelten dann solange, bis der Screen neu definiert oder geschlossen wird. Für ein Window kann keine eigene Auflösung oder Anzahl an Bit-Ebenen festgelegt werden.

Ziehen Sie bitte Ihren Workbench-Screen wieder zurück auf den Bildschirm. Geben Sie dann im BASIC-Window ein:

```
screen 1,320,200,3,1
```

Sobald Sie <RETURN> gedrückt haben, erscheint ein neuer, leerer Screen auf dem Bildschirm. Er hat eine Auflösung von 320 * 200 Punkten und bietet 8 mögliche Farben. Sie sehen, daß er ein Vordergrund/Hintergrund-Symbol hat, wie es auch bei Windows zu sehen ist. Wenn Sie das Hintergrund-Symbol anklicken, kommt wieder der Workbench-Screen in den Vordergrund. Auch in Screens kann man blättern - genau wie bei Windows.

Screens sind völlig eigenständige Bildschirme. Sie werden aber nicht auf verschiedenen Monitoren bzw. Fernsehern dargestellt, sondern auf einem einzigen. Welchen Screen Sie sehen, können Sie durch Verschieben und Blättern auswählen. Sobald ein Screen erzeugt wurde, können wir auf ihm Windows definieren. Geben Sie wieder im BASIC-Window ein:

```
window 1,"Hallo",,0,1
```

Auf unserem neuen Screen erscheint jetzt ein Window namens "Hallo". AmigaBASIC wird es gleichzeitig zum neuen BASIC-Window machen. Warum, erklären wir noch.

Auf jeden Fall gibt es jetzt zwei Screens im Amiga: Den Workbench-Screen und den Screen, den wir uns selbst definiert haben. Meistens arbeiten Sie ja auf dem Workbench-Screen, zum Beispiel, um in AmigaBASIC zu programmieren. Auf den anderen Screens sehen Sie dann die Ausgaben Ihres Programms. Damit Sie nicht immer die Vordergrund/Hintergrund-Symbole benutzen müssen, die oft durch Windows verdeckt sind, gibt es eine andere Möglichkeit, den Workbench-Screen in den Vordergrund zu holen: Die Tastenkombination <geschlossene Amiga-Taste><N> macht den Workbench-Screen zum vordersten der dargestellten Screens. Um das Gegenteil zu erreichen, drücken Sie <geschlossene Amiga-Taste><M>. Damit wird der Workbench-Screen nach ganz hinten gebracht.

Um den neuen Screen wieder zu schließen, geben Sie folgenden Befehl im BASIC-Window ein:

```
screen close 1
```

Damit haben wir allerdings den Ast abgesägt, auf dem wir saßen: Im BASIC-Window haben wir den Befehl gegeben, den Screen zu schließen, auf dem sich das BASIC-Window befindet. Jetzt haben wir keines mehr. Aber keine Sorge: Wählen Sie einfach "Show Output" aus dem "Windows"-Pulldown, und das alte BASIC-Window erscheint auf dem Workbench-Screen.

Jetzt kennen Sie schon die wichtigsten Befehle, um Screens und Windows in AmigaBASIC zu programmieren. Aber was bedeuten die ganzen Zahlen dahinter?

```
SCREEN 1,320,200,3,1
```

Die erste Zahl hinter SCREEN ist die Nummer des Screens. AmigaBASIC kann bis zu vier Screens verwalten. Deshalb sind hier Zahlen zwischen 1 und 4 erlaubt.

Die nächsten beiden Werte können Sie sich wahrscheinlich schon denken: Der erste gibt die Anzahl der horizontalen Pixels an, der zweite die vertikalen Pixels. Diese beiden Zahlen legen aber nicht die Auflösungsstufe fest, sondern lediglich die Ausdehnung des Screens innerhalb der jeweiligen Auflösung. Probieren Sie zum Beispiel mal:

```
screen 1,200,200,3,1
```

Wenn Sie für die Höhe einen Wert kleiner als 200 angeben, können beim Erscheinen des Screens zufällige Muster in den nicht vom Screen benutzten Teilen des Bildschirms erscheinen. Wenn Sie einen solchen Screen verschieben, wird er sich danach nur bis zur definierten Höhe heraufziehen lassen. Wollen Sie das mal ausprobieren?

```
screen 1,200,100,3,1
```

Der vierte Wert im SCREEN-Befehl gibt die Anzahl an Bit-Ebenen an, die der Screen haben wird. So können Sie festlegen, wieviele verschiedene Farben auf dem Screen dargestellt werden können. Die letzte Zahl schließlich gibt die Auflösungsstufe an, die für den Screen gelten soll. Dabei sind vier Werte möglich:

Wert	Auflösungsstufe	in Pixels
1	Niedrigauflösend, Normal	320 * 200 (PAL: 256)
2	Hochauflösend, Normal	640 * 200 (PAL: 256)
3	Niedrigauflösend, Interlace	320 * 400 (PAL: 512)
4	Hochauflösend, Interlace	640 * 400 (PAL: 512)

Tabelle 3: **Die möglichen Auflösungsstufen beim SCREEN-Befehl**

Die Werte, die Sie für horizontale und vertikale Ausdehnung angeben, müssen innerhalb der Grenzen der gewählten Auflösungsstufe liegen. Sonst können recht merkwürdige Ergebnisse herauskommen. Sie sollten also z.B. keine Ausdehnung von 440 * 200 Punkten wählen, wenn Sie als Auflösungsstufe nur 1

(also 320 * 200) angeben. AmigaBASIC bringt zwar keine Fehlermeldung, aber sehr sinnvoll sind solche Werte trotzdem nicht.

In der Tabelle gerade eben ist wieder der Ausdruck "Interlace" vorgekommen. Es wird Zeit, ihn zu erklären. Probieren wir zuerst aus, was der Interlace-Modus bewirkt:

```
screen 1,640,400,1,4 : window 1,"Test",(0,0)-(400,300),15,1
```

Ihnen werden nun zwei Dinge auffallen. Nämlich, daß die Buchstaben ungeheuer klein sind und daß der Bildschirm stark zittert. Das erste ist der Vorteil, das zweite der Nachteil am Interlace-Modus. Er dient dazu, eine höhere vertikale Auflösung zu ermöglichen. Tatsächlich können so auf dem PAL-Amiga bis zu 512 Pixels hohe Grafiken erzeugt werden.

Warum das möglich ist, und warum das Ganze so zittert, kann nur ein letzter Exkurs in die Fernsehtechnik erklären. Mittlerweile wissen Sie ja, daß die Sache mit den bewegten Bildern eigentlich ein Schwindel ist: Der Bildschirm stellt in rascher Reihenfolge, 25 mal pro Sekunde, ein Bild dar. Das menschliche Auge läßt sich täuschen und sieht eine Bewegung. Das ist mit 25 Bildern pro Sekunde auch problemlos möglich, allerdings flimmert das Bild dann noch erheblich. Damit man nicht mehr Bilder pro Sekunde senden muß, was die Übertragung schwieriger machen würde, benutzt man einen Trick, um das Bild stabiler darzustellen: Zwischenzeilentechnik. In der ersten Hälfte der 1/25 Sekunde stellt der Fernseher nur jede zweite Bildzeile dar. In der zweiten Hälfte wird dasselbe Bild nochmal aufgebaut, aber diesmal werden genau die Zeilen verwendet, die vorher frei geblieben waren. Das sieht für das Auge so aus, als ob 50 Bilder pro Sekunde dargestellt würden, was ausreicht, um das Bild ruhig erscheinen zu lassen.

Auch die Bilderzeugung des Amiga verwendet diesen Trick, egal ob das Bild auf einem Monitor oder einem Fernseher dargestellt wird. Damit erzeugt sie im Normalmodus eine Auflösung von 256 Bildzeilen (bzw. vertikalen Pixels).

Um diese Zeilenzahl verdoppeln zu können, sendet der Amiga im Interlace-Modus in den Zwischenzeilen ein anderes Halbbild. Die beiden versetzt dargestellten Halbbilder setzen sich für das Auge wieder zu einem kompletten Bild zusammen. Allerdings steht dieses Bild eben nicht so ruhig wie im Normalmodus, weil es jetzt nur noch mit der halben Anzahl an Bildern pro Sekunde entsteht.

Das Flimmern ist umso stärker, je höher die Kontraste zwischen den benachbarten Farben sind. Ob und wann Sie den Interlace-Modus verwenden wollen, müssen Sie selbst entscheiden. Es hängt nicht nur von den Farben, sondern auch von der dargestellten Grafik ab, wie stark das Bild zittert.

Abgesehen davon gibt es, wenn Sie aus beruflichen Gründen oder einfach nur so sehr an einem absolut ruhigen Bild interessiert sind, noch die Möglichkeit, mit einem speziellen Monitor zu arbeiten, der durch eine längere Nachleuchtdauer (für Nicht-Techniker: eine Art Verzögerungseffekt) das Flimmern mindert. Apropos Flimmern - vielleicht finden Sie dieses Flimmern reichlich unprofessionell von einem Computer wie dem Amiga. Dann machen Sie sich bitte mal die Mühe und achten Sie bei der Tagesschau oder dem Heute-Journal auf die oberen und unteren Ränder der Tafeln, die hinter dem Sprecher eingeblendet werden. Merken Sie was?

2.4 Fenster sind zum Malen da - Windows

Wenn die benötigen Screens definiert sind, besteht der nächste Schritt darin, Windows auf dem jeweiligen Screen anzulegen. Wie Sie sich nach unseren Beispielen sicher schon denken können, wird dazu der BASIC-Befehl WINDOW benutzt. Falls Sie noch den Interlace-Screen aus dem letzten Kapitel auf dem Bildschirm haben (das merkt man übrigens daran, daß auch die anderen Screens leicht zittern), löschen Sie ihn bitte mit

```
screen close 1
```

Um wieder zu einem BASIC-Window zu kommen, wählen Sie bitte "Show Output" aus dem "Windows"-Menü an. Geben Sie dann im BASIC-Window ein:

```
window 2,"Hallo",(320,20)-(600,150),31,-1
```

Daraufhin erscheint ein Window namens "Hallo" auf dem Workbench-Screen. Tippen Sie jetzt ein:

```
for x=1 to 100 : ? x : next x
```

Sie sehen, daß die ausgegebenen Zahlen in dem neuen Window erscheinen. Mit dem Befehl

```
window output 1
```

erreichen Sie, daß die Ausgaben wieder wie gewohnt ins BASIC-Window kommen.

Es wird wohl Zeit, ein wenig Systematik in die ganze Sache zu bringen. Zunächst einmal die Parameter des WINDOW-Befehls:

```
WINDOW 2,"Hallo",(320,20)-(600,150),31,-1
```

Die erste Zahl hinter dem Befehl gibt die Nummer des Windows an. Sie können in AmigaBASIC beliebig viele Windows anlegen, solange der Speicherplatz ausreicht. Die Nummer 1 ist für das BASIC-Window reserviert. Für eigene Windows sollten Sie also Nummern ab 2 verwenden. Wenn Sie ein neues Window 1 angeben, können Sie die Größe, Auflösung und Farbenvielfalt das BASIC-Windows verändern. Wollen Sie mehr als vier Farben oder eine höhere Auflösung als 640 * 256 Punkte, muß das Window allerdings auf einem neuen Screen liegen, denn die Einstellungen des Workbench-Screens können nicht verändert werden. Er ist, zumindest in den jetzigen Versionen der Workbench und von AmigaBASIC, festgelegt auf 640 * 256 Punkte Auflösung und zwei Bit-Ebenen, sprich vier Farben.

Die zweite Angabe beim WINDOW-Befehl ist der Name, den das neue Window bekommen soll. Er wird als String angegeben, das heißt, entweder als Text in Anführungszeichen oder durch eine Stringvariable.

An dritter Stelle können Sie die Lage und die Größe des Windows angeben. Fehlt diese Angabe bei der Definition, nimmt das Window die volle Größe des jeweiligen Screens an. Wollen Sie andere Werte festlegen, müssen Sie die Koordinaten der linken oberen Ecke des Windows (z.B. (100,10)) und der rechten unteren Ecke des Windows (z.B. (500,140)) in der Form (x1,y1)-(x2,y2) angeben.

Der nächste (vierte) Wert legt fest, was mit dem erzeugten Window alles gemacht werden darf und was nicht. Dazu wird die hier angegebene Zahl aus mehreren Einzelwerten zusammengesetzt, je nachdem, welche Eigenschaften das Window haben soll.

Folgende Einzelwerte sind möglich:

1	Die Größe des Windows darf mit dem Größensymbol verändert werden.
2	Das Window kann mit der Maus verschoben werden.
4	Das Window kann mit dem Vordergrund/Hintergrund-Symbol in den Vorder- bzw. Hintergrund geklickt werden.
8	Das Window kann mit dem Schließsymbol ausgeklickt werden.
16	Der Inhalt des Windows bleibt gespeichert, wenn sich die Größe ändert oder ein anderes Window im Vordergrund liegt.

<u>Tabelle 4:</u> Die möglichen Angaben beim WINDOW-Befehl

Wenn Sie zum Beispiel ein Window anlegen wollen, das bewegt werden kann (Wert 2) und ausgeklickt werden darf (Wert 8), geben Sie als vierten Wert 2+8, also 10 an. Ein Window, das in der Größe verändert (1), bewegt (2) und in den Hintergrund geklickt werden darf (4), bekommt den Wert 1+2+4, also 7. Wenn für ein Window alles erlaubt ist (1+2+4+8+16), hat es den Wert 31.

Aber Achtung: Vor allem die letzte Option (16) kostet viel Speicherplatz - denn dazu muß sich Amiga ja den gesamten Inhalt eines Fensters ständig merken, um ihn jederzeit wieder herstellen zu können. Falls der Windowinhalt nicht gespeichert bleibt (wozu Sie eben den Wert 16 angeben müssen), verschwindet er, nachdem ein anderes Window vor ihm lag oder nachdem das Window bewegt wurde. Deshalb sollte man dafür sorgen, daß ein Window, bei dem der Einzelwert 16 nicht mit angegeben werden konnte, immer im Vordergrund liegt und nicht verschoben wird. Am besten verwendet man in diesem Fall den Wert 0 (nichts ist erlaubt). Je nachdem, welchen Wert Sie eingeben, werden Sie einen Teil der üblichen Window-Symbole nicht am Windowrahmen finden.

Der letzte Wert beim WINDOW-Befehl schließlich gibt an, auf welchem Screen sich das Window befinden soll. Wird kein Wert oder die Zahl -1 angegeben, erscheint das Window auf dem Workbench-Screen.

Ein Window ist ab dem Zeitpunkt, an dem es mit dem Befehl WINDOW erzeugt wurde, automatisch das Ausgabe-Window. Das heißt, hier erscheinen die Ausgaben Ihres Programms. Um die Ausgabe auf ein beliebiges Window umzuleiten, gibt es den Befehl

WINDOW OUTPUT Window-Nummer

Probieren Sie die Befehle, die wir Ihnen vorstellen, ruhig an eigenen Beispielen aus. WINDOW OUTPUT leitet lediglich die Ausgabe um und ändert nichts an der Darstellung oder Lage der Windows. AmigaBASIC erlaubt sogar, die Ausgabe auf ein geschlossenes oder nicht sichtbares Window umzuleiten. Wenn Sie

wollen, daß das gewählte Ausgabe-Window gleichzeitig auch in den Vordergrund geholt wird, verwenden Sie einfach den WINDOW-Befehl ohne irgendwelche weiteren Werte:

WINDOW Window-Nummer

Das geht aber nur mit Windows, die vorher definiert worden sind.

Schließlich gibt es noch den Befehl

WINDOW CLOSE Window-Nummer

Er nimmt ein definiertes Window vom Bildschirm, allerdings ohne es dabei zu löschen. Das Window verschwindet zwar aus Ihrer Sicht, ist aber immer noch im Speicher! Es gibt keinen Befehl, um ein einzelnes Window, das einmal eröffnet wurde, endgültig zu löschen. Sogar der Befehl SCREEN CLOSE löscht "seine" zugehörigen Windows nicht. Nur vor dem Start mit RUN bzw. durch die entsprechende Pulldown-Option und beim Löschen eines Programms schließt AmigaBASIC alle angelegten Screens und Windows. Denken Sie daran, und verwenden Sie deshalb nicht zuviele Windows in Ihren Programmen.

Diese unkomfortable Eigenschaft von AmigaBASIC kann auch der Grund sein, wenn AmigaBASIC sich auf einmal weigert, einen WINDOW-Befehl, der "vorhin doch noch funktionierte", auszuführen. Der Fehler "Illegal function call" weist auf ein solches Problem hin, er kann aber auch andere Ursachen haben. Tritt dieser Fehler auf, sollten Sie entweder eine neue Window-Nummer verwenden, oder durch einen Neustart des aktuellen Programms erst mal alle Windows löschen.

Obwohl Sie das alles sicher an eigenen Beispielen ausprobiert haben, war's bis hierher doch etwas trocken. Mit unseren neuen Kenntnissen wollen wir deshalb gleich etwas Praktisches anfangen. Unser Videotitel-Programm läuft ja bisher nur mit vier Farben. Das können wir jetzt ändern.

Laden Sie bitte das Programm "Videotitel" und ergänzen Sie den Programmteil 'Vorbereitungen:' so, daß er wie folgt aussieht:

```
Vorbereitungen:
Farben=2
d=15 : Maxfarbe=(2^Farben)-1
Textfa=1
SCREEN CLOSE 2
IF Farben>2 THEN SCREEN 2,640,200,Farben,2 : WINDOW
2,"Videotitel",,28,2
DIM Text$(d),Farbfeld(d,3),Beweg(d),Geschw(d)
Fuell$=STRING$(16,"-")
```

In der Variablen 'Farben' geben Sie die Anzahl an Bit-Ebenen an, die verwendet werden soll. Die Standardeinstellung ist 2. Falls Sie mehr Farben benutzen wollen geben Sie 3 (für 8 Farben) oder 4 (für 16 Farben) an. Wenn der Inhalt von 'Farben' größer als 2 ist, legt das Programm einen entsprechenden Screen und ein entsprechendes Window an. Am besten können Sie das sehen, wenn Sie im Programm den Menüpunkt 4 ("Farben festlegen") wählen. Vergessen Sie nicht, die erweiterte Version gleich wieder abzuspeichen - vielleicht unter einem neuen Namen, wenn Sie die Ur-Version nicht löschen wollen.
In der Version des Videotitel-Programms, das Sie auf unserer Diskette im Buch finden, steht dieser Teil übrigens schon.

Aber das Beispiel von eben war erst der Anfang. Nachdem Sie jetzt Screens und Windows in AmigaBASIC kennen und programmieren können, wollen wir uns im nächsten Kapitel mit den Befehlen beschäftigen, mit denen wir in unsere Windows ein wenig Grafik bringen können.

2.5 Die Vielseitigen - erste Grafikbefehle

Wir wollen uns jetzt mit den Grafibefehlen des Amiga beschäftigen. In den folgenden Kapiteln wird Ihnen immer wieder eines auffallen: Hinter einem einzigen Befehl steckt oft noch eine ganze Latte von Optionen, die ihn sehr vielseitig machen.

Für unsere ersten Experimente im Direktmodus holen Sie bitte das BASIC-Window ganz in den Vordergrund, damit Sie es vollständig sehen können. Wir wollen ganz klein anfangen. Klein zumindest, was die Größe des Ergebnisses betrifft. Probieren Sie mal:

```
pset (400,100)
```

Vielleicht haben Sie den Kleinen überhaupt nicht bemerkt, aber auf dem Bildschirm ist ein weißer Punkt erschienen. Ungefähr an der Position, wo sonst das LIST-Window beginnt und etwa auf halber Höhe des Bildschirms. Wenn Sie ihn überhaupt nicht entdecken können, vergrößern wir ihn halt etwas. Mit

```
pset (401,100)
```

setzen wir unmittelbar daneben noch einen weißen Punkt. Jetzt müßte ihn aber jeder sehen können. Der BASIC-Befehl PSET wird benutzt, um einzelne Punkte in einer Grafik zu erzeugen. Da ein einzelner Punkt natürlich noch keinen allzu großen Staat macht, wollen wir den Amiga jetzt mehrere Punkte nebeneinander zeichnen lassen:

```
for x=1 to 184 : pset (x,x),3 : next x
```

Schon ist eine Linie entstanden, zur Abwechslung in oranger Farbe.

Beim PSET-Befehl geben Sie durch eine x- und eine y-Koordinate an, wo der Punkt entstehen soll. Und zwar, wie auch beim Window-Befehl, in der Form (x,y), also in Klammern. Das ganze funktioniert wie beim berühmten "Schiffe versenken". Zwei Koordinaten geben den Zielpunkt an. Nur, daß der Amiga eben nicht irgendwann sagt: "Schiff versenkt." Apropos: "Schiffe versenken" könnten Sie auf dem Amiga ja auch programmieren. Aber bevor Sie das anfangen, sollten Sie lieber noch ein bißchen Erfahrungen sammeln.

In einem normalen Window auf einem 640 * 200-Screen kann x zwischen 0 und 617 liegen, y zwischen 0 und 184. Ein Teil des

verfügbaren Platzes wird nämlich durch den Rahmen und die Symbole des Windows benötigt. Hinter den Koordinaten kann dann noch die Zeichenfarbe angegeben werden. Die vollständige Syntax lautet also:

PSET (x,y),Zeichenfarbe

Die BASIC-Zeile, die eine Linie erzeugt hat, funktioniert ganz einfach: Die Variable 'x' wird von 1 auf 184 hochgezählt. Die x-Koordinate und die y-Koordinate sollen immer denselben Wert haben, nämlich den von 'x'. So wird bei (1,1) ein Punkt gezeichnet, bei (2,2), bei (3,3) usw. bis (184,184). Das Ergebnis ist eine Linie, die in einem ganz bestimmten Winkel über den Bildschirm verläuft. Allerdings bietet AmigaBASIC eine wesentlich einfachere Methode, Linien zu zeichnen: Den Befehl LINE. Die Linie, die wir gerade noch aus Punkten gebastelt haben, wird mit LINE so programmiert:

```
LINE (1,1)-(184,184),3
```

Wenn Sie das Ergebnis sehen wollen, müssen Sie allerdings vorher den Bildschirm mit CLS löschen, sonst wird ja die neue Linie einfach über die alte gemalt und Sie sehen keinen Unterschied.

Beim LINE-Befehl geben Sie einfach den Anfangspunkt der Linie an, dann ein "bis"-Zeichen (-) und dann den Endpunkt der Linie, sowie gegebenenfalls noch die Zeichenfarbe. Sie können auch die erste Koordinate weglassen, dann beginnt die Linie an dem Punkt, wo die letzte aufhörte:

```
LINE -(500,10),2
```

Eine einfache Möglichkeit, interessante Grafiken zu erzeugen, ist die Benutzung von Zufallswerten. Dazu gibt es eine besondere Variable, sie heißt RND (kommt von engl. RaNDom, deutsch: zufällig). Die Variable RND liefert immer Zufallszahlen, die irgendwo zwischen 0 und 1 liegen. Schauen Sie sich doch mal ein paar davon an:

```
for x=1 to 10 : ? rnd : next x
```

Mit diesem Handwerkszeug können wir jetzt schon ganz hübsche Effekte erzielen. Schreiben Sie das folgende Programm bitte ins LIST-Window und speichern Sie es gleich ab. Wir werden es dann Stück für Stück verändern. Scheuen Sie sich nicht, auch eigene Experimente zu unternehmen. Wenn Ihnen ein Ergebnis besonders gut gefällt, sollten Sie das Programm gleich wieder unter einem neuen Namen abspeichern.

Die erste Fassung erzeugt farbige Punkte an zufälligen Positionen:

```
CLS
Schleife:
PSET (617*RND,184*RND),3*RND
GOTO Schleife
```

Nochmal zurück zu den Zufallszahlen: Wenn Sie RND mit einer Zahl (z.B. 617) multiplizieren, erhalten Sie eine Zufallszahl zwischen 0 und der angegebenen Zahl (z.B. 617). So werden die x-, die y-Koordinate und die verwendete Farbe festgelegt. Nach einiger Zeit ist der Bildschirm mit farbigen Punkten ziemlich übersät. Wenn Sie wollen und genug Speicherplatz haben, können Sie das Ganze ja auf einem anderen Screen mit mehr Farben haben:

```
Farben=4
SCREEN 1,640,200,Farben,2
WINDOW 2,"Punkte",,31,1
CLS
Schleife:
PSET (617*RND,184*RND),Farben*RND
GOTO Schleife
```

Sie wollen vielleicht auch noch die RGB-Zusammensetzung der Farben auf Zufallsbasis festlegen. Fügen Sie einfach vor dem CLS-Befehl noch diese Zeilen ein:

```
FOR x=1 TO (2^Farben)-1
PALETTE x,RND,RND,RND
NEXT x
```

Die nun entstandene Version finden Sie auf unserer Diskette im Buch übrigens unter dem Namen "Punkte" in der "Grafik"-Schublade. Aber zurück zum Programm:

Die Formel (2^Farben)-1 werden Sie öfter benutzen. Sie errechnet aus den vorhandenen Bit-Ebenen (in der Variablen 'Farben"') die höchste erlaubte Farbnummer. Das Zeichen ^ steht für "hoch", also potenzieren: 2^4 schreibt man in BASIC 2^4.

Beim PALETTE-Befehl ist RND ja sehr einfach zu benutzen: Die Werte für Rot, Grün und Blau können zwischen 0 und 1 liegen, und RND liefert genau so eine Zahl. Ersetzen Sie die Zeile, in der der PSET-Befehl steht, durch

```
LINE -(617*RND,184*RND),Farben*RND
```

und Sie bekommen zufällige Linien. Das Ergebnis erinnert stark an ein nicht aufgeräumtes Mikado-Spiel, finden Sie nicht? Der LINE-Befehl bietet mehr, als man zunächst vermutet. Probieren Sie im BASIC-Window folgendes:

```
LINE (10,10)-(300,100),1,b
```

Ein Rechteck ist entstanden. Auch das kann der LINE-Befehl. Na, hätten Sie ihm das zugetraut? Wie schon eingangs erwähnt: Besonders bei den Grafik-Befehlen kann man durch die Angabe von zusätzlichen Parametern oft verblüffende Effekte erzielen. Das ,b steht für Block, übersetzt also: Rechteck.

Aber woher wußte der Amiga, wie das Rechteck liegen soll? Die Anfangskoordinate legt die linke obere Ecke fest, der Endkoordinate die rechte untere Ecke. Das erinnert Sie vielleicht an die Schreibweise von Koordinaten bei Windows. Stimmt auch, denn Sie geben in AmigaBASIC die Begrenzungspunkte für Punkte, Linien, Windows, Rechtecke und ähnliches immer in der gleichen Form an.

Fügen Sie an den LINE-Befehl in unserem kleinen Beispielprogramm noch ein ,b an:

```
LINE -(617*RND,184*RND),Farben*RND,b
```

Das Ergebnis sieht wieder recht interessant aus. Und als ob's noch nicht genug wäre, bietet der LINE-Befehl noch eine weitere Überraschung: Sie können die Rechtecke auch noch ausmalen lassen. Dazu hängen Sie anstatt ,b einfach ,bf an (d.h. Block Fill, also Rechteck füllen). Probieren Sie das wieder in unserem Listing.

```
LINE -(617*RND,184*RND),Farben*RND,bf
```

Auch nicht schlecht, oder? Nun soll es aber auch Menschen geben, die nichts dem Zufall überlassen wollen und lieber alles so exakt wie möglich vorausberechnen. Wenn Sie eher zu diesem Typ gehören, konnten Sie sich vielleicht mit den Zufallsfunktionen nicht so recht anfreunden.

Ein anderer Weg, um Computergrafiken zu erzeugen, ist die Verwendung mathematischer Formeln. Auch dabei unterstützt Sie AmigaBASIC. Erinnern Sie sich zum Beispiel noch an die Sinus-Kurven aus den ungezählten Mathematik-Stunden? Speichern Sie bitte das aktuelle Programm ab, löschen Sie dann den Speicher mit NEW und geben Sie ein:

```
FOR x=0 TO 617
y=90-80*SIN(x/40)
PSET (x,y)
NEXT x
```

Wissen Sie noch entfernt, was die Sinus-Funktion bewirkt? Also, zunächst mal liefert sie Ergebnisse im Bereich von -1 bis +1. Dadurch entsteht die bekannte Kurve, die den Herstellern von kleinen Plastikschablonen enormes Geld gebracht hat. Unsere Formel in der zweiten Zeile rechnet die Ergebnisse der SIN-Funktion (so heißt sie nämlich in AmigaBASIC) auf die Bildschirm-Koordinaten um. Der x-Wert wird durch 40 geteilt, damit die Kurven nicht zu eng werden. Das Ergebnis wird mit 80

multipliziert (jetzt haben wir Zahlen von -80 bis +80) und von der Zahl 90 abgezogen. So kommen schließlich für den Bildschirm y-Koordinaten zwischen 10 und 170 raus.

Wenn Ihnen das alles zu mathematisch klang, ändern Sie einfach die eine oder andere Zahl in der Formel und schauen Sie, was passiert. (Die 40 hinten dürfen Sie kleiner oder größer machen, die 80 sollten Sie nur verkleinern und die 90 vorne bitte vorerst ganz in Ruhe lassen.)

Um anstelle der gepunkteten Linie eine durchgezeichnete Kurve zu erreichen, verwenden wir wieder den LINE-Befehl. Sie müssen ihm nur diesmal einen Startpunkt vorgeben, damit die erste Linie nicht an irgendeiner zufälligen Stelle auf dem Bildschirm anfängt. Deshalb kommt ganz vorne noch der PSET-Befehl dazu.

```
PSET (0,90)
FOR x=0 TO 617
y=90-80*SIN(x/40)
LINE -(x,y)
NEXT x
```

Wenn Sie anstelle von SIN einmal COS (= Cosinus, das ist der Kumpel vom Sinus, die beiden werden häufig zusammen angetroffen) ausprobieren, sehen Sie, daß sich die Kurve verändert. Mit SIN und COS lassen sich noch viele hübsche Effekte erzielen. Wir haben Ihnen ein paar Programmbeispiele zusammengestellt:

```
FOR x=0 TO 617
y=90-80*SIN(x/40)
LINE (0,90)-(x,y)
NEXT x
```

Wenn Sie die Punkte auf einer Sinus-Kurve mit einem festen Punkt verbinden, kommt ein interessantes Linien-Gebilde dabei heraus. Vielleicht noch das Ganze mit mehr Farbe?

```
Farben=4
SCREEN 1,640,200,Farben,2
WINDOW 2,"Sinus-Spektrum",,31,1
FOR x=0 TO 617
Fa=Fa+1 : IF Fa>7 THEN Fa=0
y=90-80*SIN(x/40)
LINE (0,90)-(x,y),Fa
NEXT x
```

Dieses Beispiel finden Sie, wie die meisten Beispiele von gerade eben auch, auf unserer Diskette im Buch. Es hat den Namen "Sinus-Spektrum"

Sie sollten auch mal andere Punkte probieren, z.B. in der LINE-Zeile:

```
LINE (320,90)-(x,y),Fa
```

oder

```
LINE (320,184)-(x,y),Fa
```

Verwenden Sie auch die Block- und Blockfill-Optionen beim LINE-Befehl:

```
LINE (320,184)-(x,y),Fa,b
```

Sie sehen, Ihrer Fantasie sind keine Grenzen gesetzt. Verstehen Sie unsere Listings wirklich nur als Beispiele und experimentieren Sie selbst! Wenn Sie etwas komplexere Formeln verwenden, kommt zum Beispiel so etwas dabei heraus:

```
Farben=4
SCREEN 1,640,200,Farben,2
WINDOW 2,"Sinusfelder",,31,1
FOR x=0 TO 617
Fa=Fa+1 : IF Fa>7 THEN Fa=0
```

```
y1=90+80*SIN(x/40)
y2=90+70*SIN(x/60)
LINE (x,y1)-(617-x,y2),Fa
NEXT x
```

Dieses Programm ist übrigens auf der beiliegenden Diskette unter dem Namen "Sinusfelder1" zu finden.

Oder auch:

```
Farben=4
SCREEN 1,640,200,Farben,2
WINDOW 2,"Sinusfelder",,31,1
FOR x=0 TO 617
Fa=Fa+.5 : IF Fa>7 THEN Fa=1
y1=90+80*SIN(x/40)
y2=90+70*SIN(x/60)
x2=320-300*SIN(x-50)
LINE (x,y1)-(x2,y2),Fa
NEXT x
```

Wir hoffen, daß wir Ihnen mit diesen Beispielen ein wenig Lust darauf gemacht haben, eigene Kreationen auszuprobieren. Vergessen Sie nicht: Schlimmstenfalls gibt's einen Error, oder das Ergebnis sieht nicht besonders gut aus. Aber im Lauf Ihrer Experimente werden Sie sicher ein Gefühl dafür entwickeln, mit welchen Mitteln Sie eindrucksvolle Grafiken erzeugen können.

Schon mal drüber nachgedacht, damit das Videoprogramm aufzupäppeln?

Zwischenspiel 3: Aus zwei mach eins - Verketten von Programmen

Vielleicht geht es Ihnen auch so: Wenn man nur lange genug mit den Sinusformeln und den LINE-Befehlen gespielt hat, hat man irgendwann eine persönliche Lieblingsgrafik, die man einfach toll findet. Es wäre doch schön, wenn man eine solche Grafik als Hintergrund im Videotitel-Programm verwenden könnte!

Denn die fliegenden Objekte und die farbigen Texte mögen ja schon ganz gut sein, aber im Hintergrund sieht es noch recht leer aus. In diesem Zwischenspiel erfahren Sie, wie sich das ändern läßt.

Es gibt verschiedene Möglichkeiten, für Hintergründe zu sorgen. Wir werden Ihnen im Verlauf des Buches auch noch einige andere Ideen zu diesem Thema vorstellen. Zunächst verwenden wir aber eine ganz einfache Methode: Wir hängen das Programm, das die Sinus-Grafik zeichnet, hinten ans Videotitel-Programm an und rufen es auf, bevor der Text und die Bewegung wiedergegeben werden. Gerade Sinus-Grafiken sehen bereits während ihrer Entstehung so interessant aus, daß es überhaupt nichts macht, wenn mit aufgezeichnet wird, wie sie sich aufbauen.

Übrigens: Wenn Sie mit unserem Programm Videotitel erzeugen, sollten Sie den Titel, sobald er fertig ist, erst einmal mit der Stopuhr in der Hand durchlaufen lassen, damit Sie wissen, wie lang er ist. Es wäre schade, wenn Sie einen Teil des Films, der dahinter auf dem Band ist, löschen würden, nur weil der Titel etwas länger dauert als gedacht.

Nun aber zu unserem Thema: Es ist ja unnötige Arbeit, wenn Sie jetzt Ihr Programm irgendwo abschreiben oder ausdrucken und dann genau dieselben Zeilen am Ende des Videotitel-Programms wieder eintippen. AmigaBASIC bietet nämlich eine Möglichkeit, Programme miteinander zu verketten, und genau das wollen wir Ihnen jetzt beibringen.

Bringen Sie zunächst das Programm in den Speicher, das die Grafik erzeugt. Entweder Sie laden es von Diskette, oder es steht schon im Speicher, weil es das letzte Programm war, mit dem Sie gearbeitet haben. Es muß jetzt in einem besonderen Format auf Diskette zurückgespeichert werden. Geben Sie im BASIC-Window ein:

```
save "Grafik",a
```

Eine unserer Sinusgrafiken finden Sie unter dem Namen "Grafik" als Beispiel in der "Videotitel"-Schublade auf der Diskette im Buch.

Das Neue bei dem Befehl von oben ist das ,a am Schluß. a steht für ASCII. Haben Sie das nicht schon einmal gehört? Richtig, ASCII-Codes waren die Codes, die bei der CHR$-Funktion angegeben werden müssen. ASCII heißt (immer noch) "American Standard Code for Information Interchange". Mit der Option ,a hinter einem SAVE-Befehl geben Sie an, daß das Programm in Form von ASCII-Zeichen auf der Diskette abgespeichert werden soll.

Wenn das so etwas Besonderes ist, wie sieht dann der Normalfall aus? Normalerweise geben Sie ja SAVE ohne eine Option ein, oder Sie wählen einfach SAVE aus dem Project-Pulldown. In diesem Fall wird das Programm in einer speziellen, speicherplatzsparenden Form auf Diskette geschrieben. (Wen's interessiert: Das Stichwort heißt *Token* und ist im Anhang D zu finden.)

Wenn sich ein Programm in dieser Form auf Diskette befindet, kann es nicht an ein anderes angehängt werden. Das geht nur, wenn Sie im ASCII-Format abspeichern. Weil wir gerade dabei sind: Es gibt noch eine Option, die Sie bei SAVE verwenden können:

```
SAVE "Test",p
```

Das kleine p steht für "protect" (deutsch: schützen). Dahinter steckt folgendes: Es gibt verschiedene Gründe, warum es nicht in Ihrem Interesse liegen könnte, daß andere Leute sich ein Programm anschauen, das Sie erstellt haben. Zum Beispiel, wenn Sie einmal Software schreiben, die Sie verkaufen wollen. Der Anwender soll es laden und starten können, aber das Programmlisting geht ihn nichts an. Zu diesem Zweck können Sie ein Programm im Protect-Format abspeichern. Es wird beim Schreiben auf Diskette verschlüsselt und kann von AmigaBASIC nicht mehr entschlüsselt werden. Nach dem Anklicken oder Laden eines solchen Programms ist das LIST-Window blockiert, und das

Programm kann nur noch gestartet werden. Auch Abspeichern ist nicht mehr möglich. Es sollte jedoch ehrlicherweise gleich dazugesagt werden, daß dieser Schutz einen professionellen Programm-Knacker nicht allzulang aufhalten dürfte. Sollte es wirklich einmal interessante Programme geben, die auf diese Weise geschützt sind, ist es sicher nur eine Frage der Zeit, wann es ein Gegen-Programm gibt, das den Schutz genauso schnell und einfach wieder entfernt. Für den Hausgebrauch ist die p-Option aber sicher nicht schlecht.

Nach diesem Abstecher in die Abgründe des Raubkopierens und Programme-Knackens wollen wir wieder zu harmloseren Anwendungen zurückkehren: Wenn Sie das Programm, das Sie anhängen wollen, abgespeichert haben, laden Sie bitte wie gewohnt unser Videotitel-Programm.

```
load "Videotitel"
```

Beim Verketten von zwei Programmen muß sich das erste Programm im Speicher befinden, das zweite wird von Diskette dazugeladen. Der Befehl, der ein im ASCII-Format abgespeichertes Programm an das aktuelle Programm anhängt, ist ganz einfach:

```
merge "Grafik"
```

Falls Ihr Programm nicht "Grafik" hieß, müssen Sie natürlich den von Ihnen verwendeten Namen angeben. "Merge" heißt auf Deutsch übrigens "verschmelzen, sich einverleiben". Damit ist alles ausgedrückt, was dieser Befehl kann.

Am Ende des Videotitel-Programms steht nun Ihr Grafik-Programm. Das war doch ganz einfach, nicht wahr? Jetzt müssen wir nur noch dafür sorgen, daß dieses neue Unterprogramm auch ausgeführt wird. Versehen Sie es dazu zunächst mit einem Label, am besten 'Grafik:'. Falls in dem angehängten Programmteil noch SCREEN und WINDOW-Befehle stehen, müssen diese gelöscht werden. Die höchste erlaubte Farbnummer kann Ihr Programm aus der Variablen 'Maxfarbe' erfahren.

Bitte achten Sie darauf, daß Sie in dem neuen Teil keine Variablen benutzen, die im Hauptprogramm benötigt werden. So etwas ist nämlich eine beliebte Ursache für Errors. Am Ende des Programms muß noch ein RETURN-Befehl eingefügt werden. In unserem Fall sieht das zum Beispiel so aus:

```
Grafik:
FOR x=0 TO 570
Fa=Fa+1 : IF Fa>Maxfarbe THEN Fa=0
y1=90-80*SIN(x2/200)
y2=90+70*SIN(x/60)
x2=320-300*SIN(x/50)
LINE (640-x2,y1)-(x2,y2),Fa
NEXT x
RETURN
```

Alles was jetzt noch fehlt, ist der Aufruf des angehängten Programms. Ihn fügen Sie bitte im Programmteil 'WiedergStart:' des Videotitel-Programms ein:

```
WiedergStart:
WIDTH 80
COLOR Textfa,Hintgr : CLS
GOSUB Grafik
COLOR Textfa,Texthi
  .
  .
  .
```

So, das war's schon. Sie können sich das Ergebnis ja gleich anschauen. Vergessen Sie bitte das Abspeichern nicht, falls Ihnen die neue Version gefällt! Auf der Diskette im Buch finden Sie eine zusammengestellte Version unter dem Namen "Videotitel+Grafik".

Mit der hier vorgestellten Methode können Sie in Zukunft Programme in beliebiger Zahl und Reihenfolge aneinanderketten. Beim Erstellen größerer Programme kann das sehr hilfreich sein.

Für Ihre Videotitel können Sie natürlich verschiedene Hintergrundgrafiken "mergen". Sie könnten sich ja zum Beispiel eine Sammlung kleiner Grafikroutinen anlegen, aus der Sie dann immer ein passendes Programm auswählen. Damit Sie dazu auch genügend Programme haben, sollen Sie wieder ein paar neue Befehle kennenlernen. Deshalb machen wir jetzt gleich weiter.

2.6 Es geht rund - weitere Grafikbefehle

Der nächste Befehl, den wir kennenlernen wollen, ermöglicht es Ihnen, Kreise zu zeichnen. Ein Beispiel, was damit möglich ist:

```
CIRCLE (320,90),160
CIRCLE (260,50),15,,,,.5
CIRCLE (380,50),15,,,,.5
CIRCLE (260,53),8
CIRCLE (380,53),8
CIRCLE (380,50),45,,.2,2.2
CIRCLE (260,50),45,,.8,2.8
CIRCLE (320,80),20,,,,.7
CIRCLE (320,110),80,,3.1,0,.2
CIRCLE (320,10),10,,4.7,1.5,.9
CIRCLE (320,15),5,,5,1.8,.7
```

Das Beispiel ist zugegebenermaßen nicht ganz leicht abzutippen. Deshalb hier mal wieder groß und deutlich der Hinweis auf die Diskette im Buch: Sie finden dieses Programm unter dem Namen "Circlesmile" in der "Grafik"-Schublade.

Wenn Sie es aber wirklich selbst abtippen wollen, dann passen Sie bitte besonders bei der Anzahl der Kommas auf!

Das Ergebnis ist, so hoffen wir, in jedem Fall die Mühe wert: Sieht doch ganz niedlich aus, nicht? Wenn Sie es selbst abgetippt haben, können Sie es ja abspeichern...

Die ganze Grafik ist nur mit dem BASIC-Befehl CIRCLE realisiert worden. Das beweist wohl schon, wie leistungsfähig er ist. CIRCLE ist wohl von allen AmigaBASIC-Befehlen derjenige,

der die meisten Optionen bietet. Das lustige Gesicht, das wir im Beispielprogramm gemalt haben, besteht aus allen möglichen Kreisen, Ellipsen und Kreisstücken. Sie alle können durch verschiedene Parameter am CIRCLE-Befehl erzeugt werden.

Die einfachste Form des CIRCLE-Befehls sieht so aus:

```
CIRCLE (x,y),Radius
```

Damit wird ein Kreis mit dem Mittelpunkt (x,y) und dem angegebenen Radius gezeichnet. Der Wert für den Radius legt den Radius in Pixels fest. Die Anzahl der Pixels gilt allerdings nur für die x-Richtung. Zum y-Radius kommen wir später. Natürlich geht es auch in verschiedenen Farben:

```
CIRCLE (x,y),Radius,Zeichenfarbe
```

Nicht nur vollständige Kreise, auch Kreisstücke können mit CIRCLE erzeugt werden. Mit den nächsten beiden Werten können Sie zwei Winkel angeben. Der erste Winkel legt den Anfang des Kreisbogens fest, der zweite Winkel das Ende:

```
CIRCLE (x,y),Radius,Zeichenfarbe,Winkel1,Winkel2
```

Die Winkel liegen zwischen 0 und 2*Pi. Das entspricht der in Amerika grundsätzlich üblichen und in Deutschland auch nicht gerade unbekannten Angabe von Winkeln als Vielfache von Pi ("Bogenmaß" nennt das der Fachmann übrigens). Ein Viertelkreis geht von 0 bis 1.57, ein Halbkreis von 0 bis 3.14. Ein Dreiviertel-Kreis von 0 bis 4.71 und ein Vollkreis von 0 bis 6.28. Für einen Vollkreis können Sie die Angaben aber auch einfach weglassen. Die folgende Abbildung hilft Ihnen, für bestimmte Kreisbögen die richtigen Winkel herauszufinden:

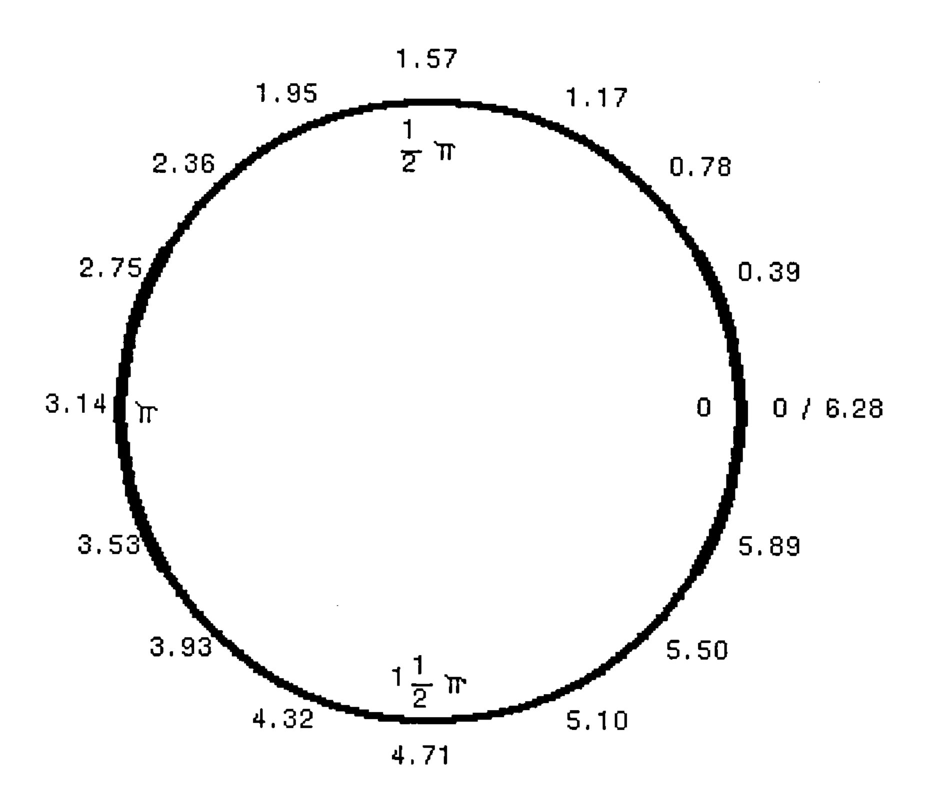


Bild 6: Kreiswinkel als Vielfache von Pi

Falls Sie bestimmte Kreissektoren (z.B. Tortenstücke) zeichnen wollen, bietet AmigaBASIC noch eine besondere Hilfe: Wenn Sie für einen Winkel eine negative Zahl angeben, wird der zugehörige Punkt am Kreis mit dem Mittelpunkt verbunden. Sie können das im BASIC-Window ausprobieren:

```
circle (320,100),200,3,-1.57,-3.14
```

Der letzte mögliche Wert schließlich (es ist der siebte mittlerweile) ist dafür zuständig, ob ein Kreis oder eine Ellipse ge-

zeichnet wird: Er gibt das Verhältnis vom x-Radius zum y-Radius an.

CIRCLE (x,y),Radius,Zeichenfarbe,Winkel1,Winkel2,x/y-Verhältnis

Will man einen hundertprozentig runden Kreis, hängt der Wert, der hier angegeben werden muß, von der Monitor-Einstellung ab. Am Amiga-Monitor 1081 können Sie mit dem linken der drei Drehknöpfe auf der Rückseite die Bildhöhe einstellen. Wenn man diese Einstellung sehr extrem vornimmt, wird schnell aus einem Kreis ein Ei oder umgekehrt. Bei normaler Einstellung des Monitors sorgt der Wert 0.44 für das richtige Verhältnis von x- und y-Radius. Die Zahl 0.44 wird deshalb auch von AmigaBASIC verwendet, wenn keine Angabe gemacht wird. Werte kleiner als 0.44 drücken den Kreis zusammen, eine liegende Ellipse entsteht. Werte größer als 0.44 ziehen den Kreis auseinander, eine stehende Ellipse (vielleicht das Ei des Kolumbus?!) entsteht.

Mit diesem Wissen gerüstet, wollen wir jetzt das Listing für das Gesicht einzeln durchgehen.

Die erste Zeile CIRCLE (320,90),160 zeichnet einen absolut runden Kreis mit dem Radius 160 Punkte um den Mittelpunkt (320,90). Das ist der Kopf.

Die nächsten beiden Zeilen CIRCLE (260,50),15,,,,.5 und CIRCLE (380,50),15,,,,.5 zeichnen die Augen: Es entstehen zwei leicht eiförmige Ellipsen, weil das x/y-Verhältnis mit 0.5 etwas über dem Normalwert liegt.

Die vierte und die fünfte Zeile CIRCLE (260,53),8 und CIRCLE (380,53),8 bringen die Pupillen in die Augen.

Mit den folgenden beiden Zeilen erzeugen wir die Augenbrauen.

```
CIRCLE (380,50),45,,.2,2.2
CIRCLE (260,50),45,,.8,2.8
```

Es werden normale Kreise mit einem Radius von 45 Pixels verwendet. Allerdings zeichnen wir bloß einen Bogen mit den Anfangswinkeln 0.2 bzw. 0.8 und den Endwinkeln 2.2 bzw. 2.8. Suchen Sie bitte diese Werte in Bild 6, um zu sehen, wo die Winkel liegen.

Jetzt ist die Nase dran: CIRCLE (320,80),20,,,,.7 - sie liegt genau im Mittelpunkt des Gesichts und hat deutlichen Ellipsen-Charakter. (Freundlicher kann das Riechorgan wohl kaum beschrieben werden...)

Für einen fröhlich lächelnden Mund nehmen wir: CIRCLE (320,110),80,,3.1,0,.2 - ein Halbkreis von Pi bis 0 (wobei der Wert 0 identisch mit 2*Pi ist), ergibt genau die untere Hälfte eines Kreises.

Um für einen, wenn auch spärlichen, Haarwuchs zu sorgen, kommen noch die letzten beiden Befehle:

```
CIRCLE (320,10),10,,4.7,1.5,.9
CIRCLE (320,15),5,,5,1.8,.7
```

Die beiden Haare sind Bögen aus einer stehenden Ellipse. Die Anfangs- und Endwinkel vergleichen Sie bitte wieder mit unserer Abbildung.

Vielleicht möchten Sie den Aufbau der Figur auf dem Bildschirm einmal selbst in dieser langsamen, schrittweisen Reihenfolge nachvollziehen? Dabei macht Ihnen der Amiga mit seiner Geschwindigkeit ja einen ziemlichen Strich durch die Rechnung. Auch der Trace-Modus kann Ihnen nicht allzuviel helfen.

Aus diesem Grund gibt es im "Run"-Pulldown eine weitere Funktion, die dazu gedacht ist, das Austesten von Programmen zu vereinfachen. Die Rede ist vom letzten Menüpunkt: "Step". Mit dieser Funktion lassen Sie ein Programm im sogenannten Einzelschritt-Modus laufen. Das heißt, daß immer nur ein Befehl ausgeführt wird, dann hält der Amiga an und wartet. Und zwar wartet er darauf, daß Sie entweder erneut "Step" im "Run"-Pulldown anwählen oder die Tastenkombinaton <geschlossene

Amiga-Taste><T> drücken. Einen BASIC-Befehl gibt es für diese Funktion nicht. Probieren Sie das doch gleich einmal aus: Jedesmal, wenn Sie "Step" anwählen oder die beiden Tasten drücken, wird genau ein Element des Gesichts gezeichnet. So können Sie Schritt für Schritt beobachten, was die einzelnen Befehle in welcher Reihenfolge bewirken. Wenn Sie das LIST-Window auf dem Bildschirm sehen, wird hier der gerade abgearbeitete Befehl genau wie bei Trace durch eine orange Umrahmung sichtbar gemacht.

Soweit alles klar? Ob schnell oder langsam, die Zeichnung erinnert bisher eher an eine Kreidezeichnung auf der Schultafel. Um sie etwas mehr Amiga-like zu machen, benutzen wir einen weiteren Befehl: Mit PAINT (deutsch: malen) können umrandete Flächen ausgemalt werden. Bitte ergänzen Sie das Programm so, daß das folgende Listing dabei herauskommt.

```
CIRCLE (320,190),200,1,,,.6
PAINT (320,170),1
CIRCLE (320,90),160,3
PAINT (320,90),3
CIRCLE (260,50),15,1,,,.5
CIRCLE (380,50),15,1,,,.5
PAINT (260,50),1
PAINT (380,50),1
CIRCLE (260,53),8,2
CIRCLE (380,53),8,2
PAINT (260,53),2
PAINT (380,53),2
CIRCLE (380,50),45,2,.2,2.2
CIRCLE (260,50),45,2,.8,2.8
CIRCLE (320,80),20,2,,,.7
CIRCLE (320,110),80,2,3.1,0,.2
CIRCLE (320,10),10,2,4.7,1.5,.9
CIRCLE (320,15),5,2,5,1.8,.7
CIRCLE (160,70),40,3,,,.6
CIRCLE (480,70),40,3,,,.6
PAINT (160,70),3
```

```
PAINT (480,70),3
CIRCLE (160,60),30,2,1,3,.3
CIRCLE (480,60),30,2,.3,2.2,.3
CIRCLE (150,65),25,2,4,1,.7
CIRCLE (490,65),25,2,2.2,5.4,.7
```

Für Tippfaule: "Circlesmile.Farbe" heißt dieses Programm auf der Diskette im Buch.

Wenn Sie selbst getippt haben, bitte gleich nach Fertigstellung abspeichern! Abgesehen davon, daß unser kleiner Mann breite Schultern und ein Paar Ohren verpaßt bekommen hat, hat er jetzt vor allem mehr Farbe im Gesicht. Wenn Sie den Aufbau der Figur genau beobachten, kommt Ihnen wahrscheinlich schon eine Idee, wie PAINT wirkt: Sie geben einen beliebigen Punkt innerhalb einer vollständig umrandeten Fläche und eine Farbnummer an. AmigaBASIC malt dann die Fläche in der angegeben Farbe aus. Die hohe Geschwindigkeit, in der das passiert, haben wir dem Blitter-Chip in der Videobaugruppe zu verdanken. Er ist besonders gut darin, Flächen zu verschieben, zu füllen oder zu verändern.

Eine Einschränkung bei PAINT ist sehr wichtig: PAINT erkennt bloß Randbegrenzungen in einer einzigen, bestimmten Farbe an. Sie haben zum Beispiel gesehen (oder wenn nicht, dann starten Sie das Programm noch mal und achten Sie darauf), daß zuerst der Körper gezeichnet und weiß ausgemalt wird, dann der Kopf erscheint und der danach folgende PAINT-Befehl den ganzen Kopf ausmalt, wobei er einen Teil des Körpers überdeckt.

Der Befehl zum Ausmalen von Flächen hat folgende genaue Syntax:

PAINT (x,y), Zeichenfarbe, Begrenzungsfarbe

Wird keine Begrenzungsfarbe angegeben, gilt die Zeichenfarbe gleichzeitig auch als Begrenzungsfarbe. PAINT (320,90),3 malt deshalb alles aus, bis es an eine orange Begrenzung stößt. Wäre

die Begrenzung z.B. weiß (Farbnr. 1) und die Füllfarbe schwarz (2), müßte folgender Befehl verwendet werden:

```
PAINT (x,y),2,1
```

Bedenken Sie diese Eigenheit des PAINT-Befehls! Sonst hat unser kleiner Mann schnell sein Gesicht verloren.

Natürlich können die Begrenzungen der Flächen, die Sie mit PAINT ausmalen, durch beliebige Grafik-Befehle entstanden sein. Die Flächen müssen nicht durch Kreise oder Ellipsen begrenzt sein. Hauptsache ist, daß die Begrenzung wirklich an allen Punkten geschlossen ist, sonst wird die Farbe auslaufen und mehr Teile ausfüllen als beabsichtigt.

Wenn Sie sich daran erinnern: Dieses Problem ist in unserem Buch schon einmal aufgetaucht, beim Object Editor, mit dem wir unsere Grafikobjekte definiert haben. Er ist ja auch ein BASIC-Programm und benutzt zum Ausmalen ebenfalls den PAINT-Befehl.

Das nächste Programm zeigt eine andere Möglichkeit, wie Begrenzungen entstehen können: Mit LINE-Befehlen wird ein Gitter gezeichnet, dessen einzelne Segmente dann zufällig ausgemalt werden. Nach einiger Zeit sieht der Bildschirm aus wie ein Spielplan beim "Schiffe versenken". Wenn wir uns das so recht überlegen: Vielleicht sollten Sie wirklich mal "Schiffe versenken" auf dem Amiga programmieren... Für die schönste Version würden wir uns vielleicht sogar eine kleine Überraschung einfallen lassen. Na, wir denken nochmal drüber nach. Jetzt aber das versprochene Listing:

```
FOR x=0 TO 615 STEP 15
LINE (x,0)-(x,185)
NEXT x
FOR y=0 TO 180 STEP 10
LINE (0,y)-(635,y)
NEXT y
```

```
Ausmalen:
PAINT (635*RND,180*RND)
GOTO Ausmalen
```

Das Programm gibt's auch auf Diskette: "Rasterfill" in der "Grafik"-Schublade.

Neu ist Ihnen vielleicht die Ergänzung STEP beim FOR...NEXT-Befehl: Damit können Sie beim Zählen eine Schrittweite angeben. FOR x=0 TO 615 STEP 15 zählt in 15er-Schritten von 0 bis 615, also: 0, 15, 30, 45, usw. So erzeugt unser Programm die notwendigen Abstände zwischen den einzelnen Gitterlinien. Ändern Sie versuchshalber die STEP-Werte, und sehen Sie, was passiert.

Jetzt kennen wir schon Befehle, um Punkte, Linien, Rechtecke, gefüllte Rechtecke, Kreise, Ellipsen und Kreisbögen zu zeichnen und einen Befehl, der umrandete Flächen ausmalt. Damit sind wir für die Grafikprogammierung ganz gut ausgerüstet. Aber AmigaBASIC bietet noch mehr Komfort beim Thema Grafik. Wie Sie ja schon wissen, kann der Blitter Flächen ungeheuer schnell füllen. Der nächste BASIC-Befehl, den wir kennenlernen wollen, macht sich das zunutze. Zuerst wieder ein Beispielprogramm:

```
COLOR 3,0
AREA (100,150)
AREA (400,150)
AREA (250,20)
AREAFILL
COLOR 2,0
AREA (100,150)
AREA (400,150)
AREA (200,180)
AREAFILL
COLOR 1,0
AREA (250,20)
AREA (400,150)
AREA (450,100)
AREAFILL
```

Dieses kleine Programm führt uns (wie seinerzeit Asterix und Obelix) in das Reich der Pyramiden. Sie sehen, wie durch drei verschiedenfarbige Dreiecke bereits ein glaubwürdiger optischer Effekt erzielt werden kann.

Auf der Diskette im Buch heißt das Programm übrigens so, wie nicht anders zu erwarten: "Pyramide".

Wir wollen jetzt aber nicht den Verdacht aufkommen lassen, AREA diene ausschließlich dazu, Dreiecksverhältnisse zu produzieren. (Das englische Wort "Area" heißt auf Deutsch übrigens "Fläche, Gebiet".) Mit diesem Befehl können Sie noch viel mehr anstellen: Sie geben mit mehreren AREA-Befehlen die Eckpunkte eines beliebigen Vielecks an. Diese Punkte speichert der Amiga solange, bis er auf den Befehl AREAFILL trifft. Und dann kommt's: Auf einen Schlag wird das angegebene Objekt auf den Bildschirm gebracht. Und das, wie bereits erwähnt, in schier unglaublicher Geschwindigkeit. Eine Einschränkung gibt's allerdings doch: Mehr als 20 Eckpunkte kann sich selbst AmigaBASIC nicht merken. Alle darüber hinaus definierten Punkte werden nicht gespeichert. Das ist aber nicht allzu tragisch, denn ein 20-Eck ist schon ein sehr komplexes Gebilde.

Wenn Sie Areas auf Zufallsbasis zeichnen lassen, kommen wieder interessante Grafiken dabei heraus. Das nächste Programm ist ein Beispiel dafür.

```
Farben=4
SCREEN 1,640,200,Farben,2
WINDOW 1,"Area-Demo",,31,1
Start:
COLOR ((2^Farben)-1)*RND,0
FOR x=1 TO 3+17*RND
AREA (617*RND,184*RND)
NEXT
AREAFILL
GOTO Start
```

Das Programm arbeitet recht einfach: Zunächst wird mit COLOR eine zufällige Farbe innerhalb des erlaubten Bereichs ausgewählt. Die FOR...NEXT-Schleife wird zwischen 3 und 20 mal durchlaufen. Der Mindestwert ist deshalb drei, weil drei Punkte nötig sind, um eine sichtbare Fläche zu erzeugen. Zwei Punkte ergäben logischerweise nur einen Strich in der Landschaft. Wenn die Fläche gezeichnet ist, springt das Programm zum Label 'Start:' zurück und beginnt den ganzen Ablauf von vorne.

Da alle Eckpunkte aus Zufallszahlen errechnet werden, kommen natürlich oft sehr verwinkelte, unförmige Gebilde dabei heraus. Aber gerade die können besonders interessant aussehen. Natürlich läßt sich der AREA-Befehl genauso gut im Rahmen mathematischer Formeln einsetzen:

```
Farben=4
SCREEN 1,640,200,Farben,2
WINDOW 1,"Areas",,31,1
FOR x=0 TO 617
Fa=Fa+1 : IF Fa>(2^Farben)-1 THEN Fa=0
COLOR Fa,0
y1=90+80*SIN(x/40)
y2=90+70*COS((617-x)/25)
x2=ABS(320-x)
AREA (x,y1)
AREA (x2,y2)
AREA (x,x/4)
AREA (320,90)
AREAFILL
NEXT x
```

Dieses Beispiel finden Sie auf unserer Diskette unter dem Namen "Areasinus".

Mit AREA und AREAFILL haben wir jetzt alle wichtigen Grafikbefehle durchgesprochen. In den folgenden Kapiteln wird es darum gehen, diese Befehle in etwas umfangreicheren Programmen einzusetzen, von denen Sie neben dem Lerneffekt vor allem einen praktischen Nutzen haben sollen. Für jeden Geschmack ist etwas dabei: Zuerst machen wir ein wenig Statistik,

das wird die eher geschäftlich/wirtschaftlich Interessierten unter Ihnen freuen. Danach kommt ein Bonbon für diejenigen, die sich mehr zu künstlerischen Anwendungen hingezogen fühlen. Wobei sich die beiden genannten Dinge dank Amiga ja nicht unbedingt ausschließen müssen. Doch nun genug der Ankündigungen, fangen wir an.

2.7 Die Wende - das Balken- und Tortengrafik-Utility

Sicher haben Sie schon einmal irgendwo Balken- und Tortengrafiken gesehen, auch wenn Ihnen die Namen vielleicht nicht allzuviel sagen. Wenn Sie zum Beispiel die letzte Wahl im Fernsehen verfolgt haben, hat Ihnen ein freundlicher Herr im Lauf des Abends immer wieder mal die aktuelle Hochrechnung in grafischer Form präsentiert. Auch Statistik ist ein beliebtes Anwendungsgebiet für Computergrafik.

Die sogenannten Tortengrafiken sind die runden Kuchen, von denen die Parteien am liebsten immer das größte Stück abbekommen wollen. Die Größe des jeweiligen Sektors entspricht dem prozentualen Anteil einer Partei an der Gesamtzahl aller abgegebenen Stimmen, nicht, wie viele mittlerweile glauben, dem Anteil vom Parteispendenkuchen.

Wahlen sind aber nicht die einzige Gelegenheit, wo Tortengrafiken eingesetzt werden. Wann immer es um Anteile (z.B. Marktanteile), Zusammensetzungen (seien es nun Parlamente, Aufsichtsräte oder die neueste Cola-Rezeptur) oder Verhältnisse geht (z.B. Gewinn verglichen mit Gesamtumsatz), eignen sich Tortengrafiken dazu, nüchterne Zahlen besser vorstellbar zu machen.

Eine andere Art, statistische Daten optisch aufzubereiten, heißt Balkengrafiken. Das sind bei der Wahlberichtserstattung die verschieden hohen Säulen, die den abgegebenen Stimmen pro Partei entsprechen.

Balkengrafiken sind hilfreich, um sich das Verhältnis verschiedener Werte zueinander deutlich zu machen. Oder die Entwicklung eines einzelnen Wertes. Man kann also z.B. Verkaufszahlen, Preisentwicklungen oder Aktienkurse mit Hilfe von Balkengrafiken darstellen. Und zwar entweder nach dem Motto "Ich gebe jetzt mal unsere Verkaufszahlen der letzten Monate ein" oder "Ich vergleiche unsere Zahlen dieses Monats mit denen der Konkurrenz".

Die Anwendungsmöglichkeiten sind so vielschichtig wie die Daten, die den Auswertungen zugrundeliegen.

Natürlich können Sie Balken- und Tortengrafiken auch auf Ihrem Amiga erzeugen. Und damit Ihnen das ein bißchen leichter fällt, haben wir für Sie ein Programm geschrieben, das Ihnen die Arbeit abnimmt. Hier zunächst das Listing:

```
Grafik:
  IF Wert(0)=0 THEN RETURN
  IF UCASE$(Wert$(0))="B" THEN GOSUB Balkengrafik
  IF UCASE$(Wert$(0))="T" THEN GOSUB Tortengrafik
RETURN

Tortengrafik:
  Gesamt=0
  FOR x=1 TO Wert(0)
    Gesamt=Gesamt+Wert(x)
  NEXT x
  Divi=Gesamt/6.283 : Winkel1=.0001 : Bfarb=1
  FOR x=1 TO Wert(0)
    Zfarb=Bfarb
    IF Zfarb>(2^Farben)-1 THEN Zfarb=1
    Bfarb=Zfarb+1
    IF Bfarb>(2^Farben)-1 THEN Bfarb=1
    Winkel2=Winkel1+Wert(x)/Divi
    CIRCLE (320,100),156,Bfarb
    CIRCLE (320,100),150,Bfarb,-Winkel2,-Winkel1
    PAINT (320,32),Zfarb,Bfarb
    CIRCLE (320,100),150,Bfarb,-Winkel1,-Winkel2
    Mitwinkel=(Winkel1+Winkel2)/2
```

```
    px=320+165*COS(Mitwinkel)
    py=100-80*SIN(Mitwinkel)
    Abstand=0
    IF Mitwinkel>1.57 AND Mitwinkel<4.72 THEN Abstand=LEN(Wert$(x))
    IF Abstand>15 THEN Abstand=15
    COLOR Zfarb,0
    LOCATE (py/8.3)+1,(px/8)+1-Abstand
    PRINT Wert$(x);
    Winkel1=Winkel2
  NEXT x

  CIRCLE (320,100),156,0
RETURN

Balkengrafik:
  Max=.0001 : Zfarb=0
  FOR x=1 TO Wert(0)
    IF Wert(x)>Max THEN Max=Wert(x)
  NEXT x
  Breite=INT(550/(Wert(0)))
  IF Breite>100 THEN Breite=100
  Fakt=160/Max
  LOCATE 2,1 : PRINT Max;
  LOCATE 12,1 : PRINT Max/2;
  FOR x=0 TO 10
    LINE (1,170-x*16)-(5,170-x*16)
  NEXT x
  FOR x=1 TO Wert(0)
    Zfarb=Zfarb+1 : IF Zfarb>(2^Farben)-1 THEN ZFarb=1
    LINE (30+(x-1)*Breite,170-Wert(x)*Fakt)-(25+x*Breite,170),Zfarb,bf
    COLOR Zfarb,0
    LOCATE 23,(4+(x-1)*(Breite/7.8))
    PRINT Wert$(x);
  NEXT x
RETURN
```

Weia, hoffentlich haben wir Ihnen da nicht zuviel zugemutet. Das ganze Ding abtippen... Aber Sie finden's natürlich auch wieder auf unserer Diskette. Es ist das Programm "Balktort" in

der "Grafik"-Schublade. Diese Version ist jedoch allein nicht lauffähig. Sie erhalten nach RUN einen "Return without Gosub"-Error. Das ist auch kein Wunder, denn der oben abgedruckte Teil ist nur ein Unterprogramm. Was bisher fehlt, ist das Hauptprogramm, das diesen Teil aufruft. Dazu kommen wir jedoch gleich. Jetzt wollen wir erstmal das Unterprogramm erklären:

Zuerst ist Ihnen sicher aufgefallen, daß das Listing rein äußerlich etwas anders aussieht als die bisherigen: Die meisten Zeilen sind ein paar Positionen eingerückt. Was hat das zu bedeuten?

Das Stichwort heißt "strukturierte Programmierung". BASIC-Programme haben nämlich oft die Eigenschaft, sehr unübersichtlich auszusehen. Meist sind sie dann auch sehr unübersichtlich programmiert. So kommt es, daß sich selbst derjenige, der das Programm geschrieben hat, nach einiger Zeit nicht mehr auskennt. Da AmigaBASIC alle Möglichkeiten bietet, die Übersichtlichkeit in Programmen zu gewährleisten (Labels, Unterprogramme etc.), sollte man dem auch optisch Rechnung tragen. Wer herausfinden will, wie ein Programm funktioniert, tut sich bei dem gerade eingetippten Listing sicher leichter als bei den Programmen, wo die Zeilen einfach auf einer Höhe untereinander stehen.

Diese Technik haben wir es hier zum ersten Mal eingesetzt, weil Sie hier auch zum ersten Mal ein ganzes Programm abtippen mußten, ohne zwischendrin logische Pausen durch unsere Erklärungen zu haben. Die Regel, wie man Programmstrukturen richtig kennzeichnet, ist ganz einfach: Wann immer ein Teil innerhalb einer Schleife oder eines Unterprogramms bzw. eine eigene logische Einheit beginnt, rücken wir den Zeilenanfang um eine oder zwei Stellen nach rechts. Ist der betreffende Teil dann zu Ende, rücken wir wieder um die gleiche Stellenzahl nach links. Das war's schon. Wer noch mehr tun möchte, kann zwischen einzelnen Programmteilen noch Leerzeilen einfügen.

Auf jeden Fall sollten Sie bitte auch diesmal das Programm auf Diskette abspeichern. Zumindest dann, wenn Sie es selbst eingetippt haben. Aber daran haben Sie in diesem Fall sicher schon selbst gedacht.

So wie das Programm jetzt im Speicher steht, können Sie es noch nicht benutzen. Es ist ja schließlich nur ein Unterprogamm, und wir haben noch kein Programm, das es aufruft. Für die ersten Experimente mit unseren Grafiken gibt es deshalb noch ein kleines Testprogramm, das Sie vor Ihrem Balken- und Tortengrafik-Utility eingeben können. Bringen Sie also bitte den Cursor an den Programmanfang, und geben Sie dort noch die nächsten Zeilen ein:

```
Vorbereitungen:
  Farben=3
  SCREEN 4,640,200,Farben,2
  WINDOW 99,"Grafik",,8,4
  PALETTE 7,.8,.2,.1
  DIM Wert(50),Wert$(50)

Testeingabe:
  WINDOW 1
  INPUT "B=Balken, T=Torten: ",Wert$(0)
Eingabe:
  Wert(0)=Wert(0)+1
  PRINT Wert(0)". Wert:";
  INPUT Wert(Wert(0))
  PRINT "Bezeichnung dazu:";
  INPUT Wert$(Wert(0))
  IF Wert$(Wert(0))<> "" THEN Eingabe

Aufruf:
  Wert(0)=Wert(0)-1
  WINDOW 99
  GOSUB Grafik
  END
```

Wenn Sie das entstandene Programm jetzt abspeichern, wählen Sie bitte einen neuen Namen, damit Sie sowohl die Test-Version als auch das reine Unterprogramm auf Diskette haben. Auf unserer Diskette im Buch hat diese erweiterte Version den Namen "Balktort.Test".

Wer will, kann das Programm gleich ausprobieren. Zunächst müssen Sie entscheiden, ob Sie mit Ihren Daten eine Balken- oder eine Tortengrafik erzeugen wollen. Geben Sie "b" für Balkengrafik ein bzw. "t" für Tortengrafik und drücken Sie <RETURN>.

Nun fragt Sie das Programm nach dem ersten Wert. Hier können Sie eine beliebige Zahl eingeben, zum Beispiel 850. Nach der Bestätigung mit <RETURN> fragt das Programm nach der Bezeichnung zu dem eben eingegebenen Wert. Diese Bezeichnung wird dann unter die Balken bzw. neben die Tortensegmente gedruckt. Sie könnten zum Beispiel "Amiga" eintippen. Im laufenden Monat sind nach Ihren Angaben 850 Amigas verkauft worden. Die nächste Zahl könnte 400 sein, Bezeichnung "Atari". Eine dritte Zahl soll den Rest aller Computermarken in sich aufnehmen: 1200, Bezeichnung: "sonstige". Danach drücken Sie bitte zweimal <RETURN>, ohne irgendwelche Werte einzugeben.

Mit diesen Eingaben können Sie jetzt schon eine aussagekräftige Grafik zum Thema Computer-Verkäufe erstellen. Für die statistische Qualität unserer Beispielzahlen können wir allerdings keine Haftung übernehmen.

Nun zur Bedienung und Funktion unseres Utilities. Ach ja, "Utility" haben wir als Begriff bisher auch noch nicht erklärt. Das englische Wort bedeutet "etwas Nützliches, Nutzbringendes" und steht im Zusammenhang mit Computern für Programme, die bestimmte Hilfen anbieten. Wir helfen Ihnen zum Beispiel, Ihre Daten in Grafik umzuwandeln, und das ist doch ganz nützlich, oder?

Das Unterprogramm übernimmt von Hauptprogramm drei verschiedene Arten von Daten. Erstens braucht es die Anzahl der erlaubten Bit-Ebenen. Wie schon gewohnt, legen wir diesen Wert in der Variablen 'Farben' ab.

Die Zahlen, die der Grafik zugrunde liegen, stehen im Datenfeld 'Wert(x)'. Dieses Feld kann beliebig viele Elemente haben, mehr als 50 sollten Sie aber mit Rücksicht auf die Übersichtlichkeit

der Grafiken auf keinen Fall verwenden. Die Anzahl der Daten, die in der Grafik dargestellt werden sollen, steht in der Feldposition 'Wert(0)'. Befindet sich hier also z.B. die Zahl 5, werden die Zahlen aus 'Wert(1)', 'Wert(2)',... bis 'Wert(5)' verwendet. Sie können für beide Grafiktypen beliebige Zahlen angeben. Das bedeutet, auch die Werte für Tortengrafiken müssen zusammen nicht 100 % ergeben, denn um die Umrechnung auf prozentuale Anteile kümmert sich das Unterprogramm selbst.

Die Bezeichnungen zu den Werten stehen im Feld 'Wert$(x)'. Für jede Zahl im Feld 'Wert(x)' steht die Bezeichnung an der entsprechenden Position in 'Wert$(x)'. Der Text zu 'Wert(1)' steht also in 'Wert$(1)' usw. In 'Wert$(0)' geben Sie durch einen Kennbuchstaben an, welche Darstellung gewünscht ist. Für Balkengrafiken muß hier ein "b" stehen, für Tortengrafiken ein "t". Bei allen anderen Buchstaben springt das Programm zurück, ohne eine Grafik zu zeichnen.

Für diese aufgeführten Daten müssen Sie also in Ihrem eigenen Programm selbst sorgen. Zur Verdeutlichung hier noch einmal alle übergebenen Variablen und ihre Werte, wie sie in unserem Beispiel verwendet wurden:

Name	Inhalt z.B.	Bemerkung
Farben	4	Anzahl Bit-Ebenen, für Workbench-Screen: 2
Wert(0)	3	Anzahl der Daten in den Feldern 'Wert' und 'Wert$'
Wert(1)	850	1. Zahl
Wert(2)	400	2. Zahl
Wert(3)	1200	3. Zahl
Wert$(0)	t	b=Balkengrafik t=Tortengrafik
Wert$(1)	Amiga	Bezeichnung zur 1. Zahl
Wert$(2)	Atari	Bezeichnung zur 2. Zahl
Wert$(3)	sonstige	Bezeichnung zur 3. Zahl

Tabelle 5: Die Übergabe-Variablen im Balken-/Tortengrafik-Utility

Mehr Angaben braucht das Unterprogramm nicht. Reicht ja auch. Wenn Sie ein eigenes Datenverarbeitungsprogramm schreiben, müssen Sie gegebenenfalls den Screen und das Window, in dem Sie die Grafik gern hätten, selbst anlegen. Vor dem Aufruf des Unterprogramms muß das gewünschte Window mit

WINDOW Windownummer

zum Ausgabe-Window gemacht werden. Dann rufen Sie das Unterprogramm mit

```
GOSUB Grafik
```

auf. Dahinter können dann wieder eigene Programmteile folgen.

Das war alles, was Sie zur Anwendung des Utilities wissen müssen. Wir besprechen jetzt noch, wie das Programm funktioniert. Wenn das stellenweise etwas zu mathematisch wird: Um das Programm zu benutzen, müssen Sie ja nicht alles ganz genau verstanden haben.

Das Testprogramm gibt im Teil 'Vorbereitungen:' zuerst die Bit-Ebenen an, legt einen 640 * 200-Screen mit der gewünschten Farbenzahl an und eröffnet ein Window mit der Nummer 99. Diese hohe Nummer haben wir gewählt, damit Ihnen das Window nicht in die Quere kommt, falls Sie unsere Test-Version selbst etwas erweitern wollen.

Der PALETTE-Befehl legt eine Farbe fest. Natürlich können Sie gern alle verwendeten Farben ändern. Beachten Sie dabei, daß PALETTE immer nur auf dem Screen wirkt, auf dem das aktuelle Ausgabe-Window liegt. Wir haben die Farbe Nummer 7 umdefiniert, weil diese beim Einschalten des Amiga die gleiche Einstellung hat wie Farbe Nummer 1 (in beiden Fällen weiß). Kommt bei einer Tortengrafik aus 7 Elementen der Sektor mit der Farbe 7 neben den Sektor mit der Farbe 1, läge weiß neben weiß und das ist etwas unpraktisch. Deshalb haben wird die Farbe 7 in Rot geändert.

Als nächstes werden die Felder 'Wert' und 'Wert$' dimensioniert, je auf 50 Elemente. Im Teil 'Testeingabe:' können Sie solange Werte in die beiden Felder eingeben, bis bei einer Bezeichnung eine leere Eingabe erfolgt. Dieses Kriterium zeigt dem Programm an, daß jetzt alle Werte vorliegen.

Der Teil 'Aufruf:' berichtigt den Inhalt von 'Wert(0)', der die Anzahl der vorhandenen Daten angibt. Dann wird Window 99 zum Ausgabe-Window gemacht und das Grafik-Unterprogramm aufgerufen. Nach der Rückkehr aus diesem Unterprogramm ist das Programm zu Ende.

Das Unterprogramm beginnt mit dem Teil 'Grafik:' und checkt zuerst ab, ob überhaupt irgendwelche Werte vorhanden sind. Falls nicht, springt es gleich zurück. Anhand des Inhalts von 'Wert$(0)' entscheidet sich nun, ob eine Balken- oder eine Tortengrafik gezeichnet wird. Wie schon einmal, macht uns die Funktion UCASE$ davon unabhängig, ob der Kennbuchstabe in Klein- oder Großschrift angegeben wurde. Innerhalb des Unterprogramms wird nun wieder ein Unterprogramm aufgerufen, nämlich das, das die gewünschte Grafik erzeugt.

Kommen wir zum Teil 'Tortengrafik:'. Die ersten vier Zeilen addieren alle vorhandenen Werte und errechnen so die Gesamtsumme. Diese Gesamtsumme entspricht 100%. Daher wird in der Variablen 'Divi' die Zahl errechnet, durch die die Zahl geteilt werden muß, um die Einzelwerte auf die zugehörigen Kreiswinkel umzurechnen. Die Zahl 6.283 ist 2*Pi, also der höchste mögliche Winkel.

Der Anfangswert von 'Winkel1' wird etwas über 0 gesetzt, weil der CIRCLE-Befehl (wie jeder gute Mathematiker (Da staunen Sie aber, Herr Ott, was? (Herr Ott war unser Mathematiklehrer))) nicht zwischen +0 und -0 unterscheidet. Bei einem Winkel von 0 würde der erste Sektor nicht ordnungsgemäß mit dem Kreismittelpunkt verbunden. 'Bfarb', die Begrenzungsfarbe, bekommt den Startwert 1 (weiß).

Die nachfolgende FOR...NEXT-Schleife beinhaltet die Zeichenbefehle. Sie wird so oft ausgeführt, wie Werte vorhanden sind. Zu Beginn der Schleife bekommt die Zeichenfarbe 'Zfarb' den alten Wert von 'Bfarb'. Falls eine der Farben über der erlaubten Höchstzahl liegt, wird sie auf 1 zurückgesetzt. Die Hintergrundfarbe 0 verwenden wir nicht zum Zeichnen. Danach wird auch die Begrenzungsfarbe um 1 erhöht.

Was hat es nun auf sich mit den beiden Farben? Wie Sie sicher noch wissen, akzeptiert PAINT zum Ausmalen von Flächen immer nur Begrenzungen in einer bestimmten Farbe. Wir wollen die Sektoren unserer Tortengrafik in verschiedenen Farben ausmalen. Dazu brauchen wir eine Zeichenfarbe, eben 'Zfarb' und eine Begrenzungsfarbe, eben 'Bfarb'.

Der Trick, mit dem wir das Ausmalen erreichen, ist nicht gerade von mathematischer Brillanz, aber recht praktisch - und das geht manchmal vor. Das Problem: PAINT benötigt ja immer einen Punkt innerhalb der Fläche, die ausgemalt werden soll. Da die Sektoren sehr unterschiedliche Größen haben können, ist dieser Punkt nicht einfach zu berechnen. Die Lösung: Zum Zeichnen der Tortengrafik werden zwei ineinanderliegende Kreise verwendet, der äußere ist ein kleines Stück größer (Radius: 156) als der innere (Radius: 150). Mit der Begrenzungsfarbe werden die Teile des Kreises geschützt, die bereits fertig sind. Dann wird die Fläche zwischen den beiden Kreisen mit PAINT ausgemalt. Das Resultat: Die Farbe läuft auch in die ungeschützten Teile des Kreises und malt diese in der angegebenen Zeichenfarbe aus. Deshalb verwenden wir für die Begrenzung immer die Farbe, die eine Nummer höher ist als die Zeichenfarbe. Ganz am Schluß wird dann der äußere Kreis gelöscht. Wenn Sie ganz genau aufpassen, können Sie dieses Prinzip während des Entstehens der Grafik verfolgen.

Schauen wir uns nun an, wie im Programm unser theoretischer Plan in die Praxis umgesetzt wurde. Die fünfte Zeile innerhalb der Schleife errechnet den neuen Wert von 'Winkel2'. Das ist der größere der beiden Begrenzungswinkel. Seine neue Größe ist der Anfangswinkel plus dem Anteil, den der nächste darzustellende Wert am Kreisbogen hat.

Zuerst wird der äußere Kreis in der aktuellen Begrenzungsfarbe gezeichnet. Dann ebenfalls in der Begrenzungsfarbe der Kreisausschnitt, der die bereits fertigen Teile schützt. (Dazu wird einfach ein Rand um den Kreis gezeichnet und dabei genau der Teil ausgespart, der ausgemalt werden soll.) Der PAINT-Befehl in der nächsten Zeile malt in der Zeichenfarbe 'Zfarb' aus und erkennt als Begrenzung die Linien, die in der Farbe 'Bfarb' gezeichnet wurden. Sein Anfangspunkt liegt genau in der freien Fläche zwischen den ineinanderliegenden Kreisen. Nach dem Ausmalen wird der gesamte innere Kreis mit der Begrenzungsfarbe umschlossen und die Fläche zwischen den Kreisen durch Ausmalen in der Hintergrundfarbe 0 gelöscht. Zu guter Letzt grenzen wir die Sektoren mit der Begrenzungsfarbe voneinander ab.

Was nun noch innerhalb der Schleife an Befehlen steht, dient hauptsächlich einem Zweck: Die zugehörigen Texte sollen neben die Sektoren gedruckt werden. Auch dafür mußten wir uns wieder einen Trick einfallen lassen: Wir rechnen zuerst den Winkel aus, der genau durch die Mitte des aktuellen Sektors läuft. Dieser Winkel soll 'Mitwinkel' heißen.

Um ehrlich zu sein: Jetzt wurde es echt knifflig. Wie um alles in der Welt kann man aus einem bekannten Radius und einem bekannten Winkel den Punkt bestimmen, wo der Winkel den Kreisbogen schneidet? Den brauchen wir nämlich, um den Text richtig zu positionieren. Wer sich das - wenn schon überhaupt - optisch besser vorstellen kann, dem zeigt die Abbildung 7, was gemeint ist:

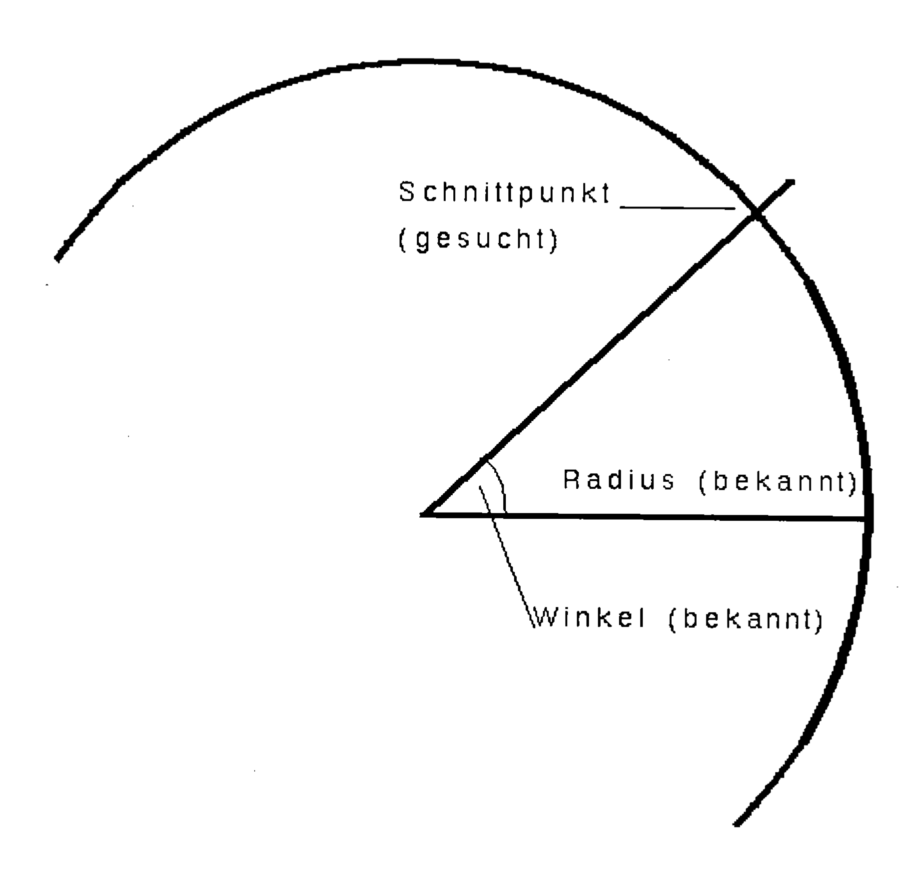


Bild 7: **Der Winkel schneidet den Kreisbogen - wir suchen den Schnittpunkt**

Wir wollen uns nicht mit fremden Federn schmücken. Die Erleuchtung kam Hannes Rügheimer nach einiger Zeit frustrierten Blätterns in der alten Formelsammlung, die ihm schon während seines Mathe-Abiturs als Schutzschild diente. Da fand sich nämlich unter der vielsagenden Überschrift "Goniometrie - Veranschaulichung am Einheitskreis" ein genialer Zusammenhang:

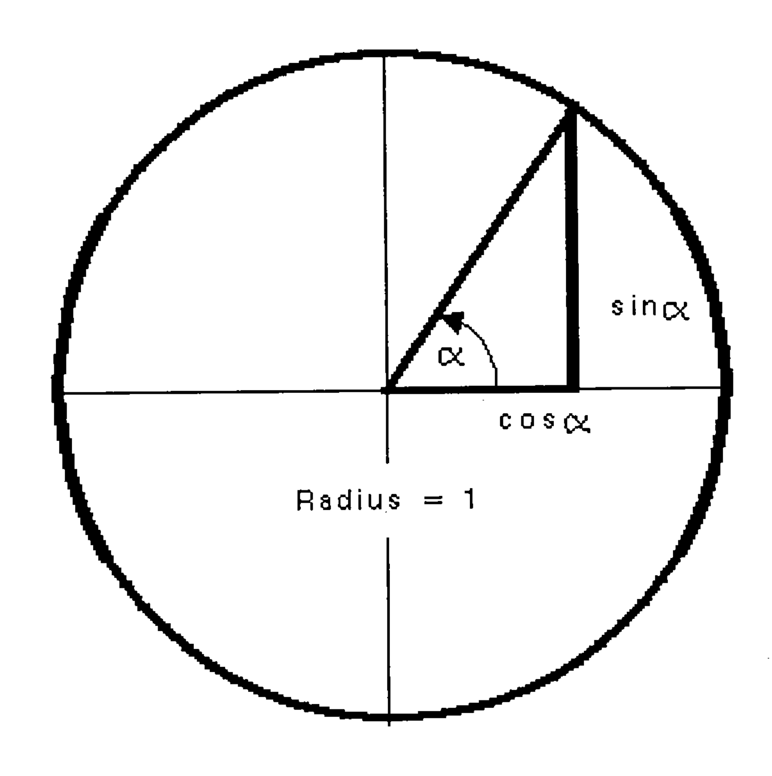


Bild 8: Sinus und Cosinus im Kreis

Sieh an! Sinus und Cosinus kann man also nicht nur für hübsche Computergrafiken gebrauchen, sondern auch für ganz nüchterne Berechnungen: Der Radius multipliziert mit dem Cosinus des Winkels ergibt die x-Koordinate des gesuchten Schnittpunkts, Radius mal Sinus des Winkels liefert die y-Koordinate. Na, das ist doch was. Nur noch auf Bildschirmkoordinaten umrechnen (der Mittelpunkt ist (320,100), und schon haben wir die Variablen 'px' und 'py'.

Das nächste Problem... Schon wieder ein Problem! Jetzt lernen Sie die wahren Abgründe des Programmierens kennen. Ähnlich wie Indiana Jones beim Weg zum Tempel des Todes kämpft sich ein Programmierer ständig mit einem symbolischen Buschmesser

in der Hand durch das Dickicht des BASIC-Dschungels. Trotzdem ist das hier nicht das Dschungel-Buch. Also gehen wir, zusammen mit Indiana Hannes, zum nächsten Problem über: Um es gleich vorwegzunehmen - wir können das Zähneklappern der Nichtmathematiker bis hierher hören - diesmal war es vergleichsweise einfach zu lösen.

Wenn sich ein Text neben der rechten Hälfte der Tortengrafik befindet, darf er beliebig lang sein. Schlimmstenfalls wird er halt am rechten Bildschirmrand abgeschnitten. Aber ein Text, der an der linken Seite steht, wird schon nach wenigen Zeichen in die Grafik hineinragen - ein Zustand, den wir nicht für wünschenswert hielten. Die Zeile, die mit dem Befehl

```
Abstand=0
```

beginnt, schafft Abhilfe bzw. den nötigen Abstand: Wenn der Winkel zwischen 1.57 (½ * Pi) und 4.72 (1½ * Pi) liegt, befindet sich der Text links von der Torte. Ist dies der Fall, wird von der Anfangsposition noch die Länge des Textes abgezogen. So wird der letzte Buchstabe der Bezeichnung genau neben dem Kreis stehen.

In diesem Zusammenhang gibt es auch mal wieder einen neuen BASIC-Befehl: LEN(a$). Er liefert die Länge eines Strings in Zeichen.

```
LEN("Hallo")
```

ist also 5 und

```
LEN("Oberpostdirektionsbriefmarkenstempelautomatenmechaniker")
```

ist 55. Sie können ja nachzählen...

Ist der Abstand des Textanfangs vom Kreis größer als 15 Zeichen, muß er allerdings auf 15 begrenzt werden, mehr Platz gibt es leider nicht auf dem Bildschirm. Schließlich werden die Koordinaten 'px' und 'py', die ja Pixels vertreten, noch in Zeilen und Spalten für LOCATE umgerechnet und der Text zu guter

Letzt auf den Bildschirm gebracht. Der gesamte Formelapparat bezieht sich übrigens auf 80 Zeichen pro Zeile. Bitte stellen Sie nur diesen Wert als Grundwert in Preferences ein.

Für den nächsten Schleifendurchlauf bekommt der Anfangswinkel ('Winkel1') den Wert des Endwinkels ('Winkel2'), und die ganze Prozedur beginnt für den folgenden Sektor von vorn.

Sind alle Sektoren gezeichnet, wird, wie schon erwähnt, noch der äußere Kreis gelöscht. Dann springt das Unterprogramm mit RETURN an die Stelle zurück, von der es aufgerufen wurde.

Hoffentlich raucht Ihnen der Kopf nicht zu sehr. Sie wissen ja: Wenn Ihnen die letzte Nuance mathematischer Spitzfindigkeiten in diesem Abschnitt entgangen sein sollte, macht's auch nichts. Zum Thema Tortengrafik bleibt uns nur noch zu wünschen: Guten Appetit.

Im Unterprogramm 'Balkengrafiken:' haben wir mit weniger erbittertem Widerstand von seiten der Logik zu rechnen. Die ersten vier Zeilen suchen aus allen Werten den Größten. Nach diesem Wert richtet sich die Höhe der anderen Balken. Er wird in die Variable 'Max' geschrieben. Wir dachten bei der Namensgebung übrigens weniger an Max und Moritz, sondern mehr an einen Maximalwert. Mindestwert für 'Max' ist übrigens deshalb 0.0001, weil wir schon in der übernächsten Zeile 550 durch Max teilen werden. Und Teilen durch 0? "Das geht doch nicht." Bravo, wer's zuerst gesagt hat, bekommt einen Punkt. Und wenn Sie damit Ihre Chance gekommen sehen, Amiga endlich einmal reinzulegen, probieren Sie im BASIC-Window:

```
? 1/0
```

Der darauf erfolgte "Division by Zero"-Error heißt übrigens auf Deutsch "Division durch Null". Sie sehen: Irgendwo in den Entwicklungsbüros von Microsoft muß zumindest ein Mathematiklehrer rumlaufen, der auf so etwas geachtet hat. Kunststück: Es gibt ja genug davon - zumindest aus der Sicht eines Schülers.

Die Variable 'Breite' gibt später die Breite unserer Balken an. Der Wert ist abhängig von der Anzahl der Balken, die dargestellt werden sollen. Aber größer als 100 Pixels sollte er auf keinen Fall werden, schließlich wollen wir normale Balken zeichnen und keine kalifornischen Tausendjahres-Eichen. Ein Balken, der die ganze Bildschirmbreite ausfüllt, kann ja nicht mehr als Balken bezeichet werden. Höchstens als Brett vorm Schirm.

'Fakt' ergibt den Wert, mit dem wir die einzelnen Werte multiplizieren müssen, um sie alle auf die richtige Größe umzurechnen. 160 Pixels hoch darf unser Maximalwert sein, alle anderen werden entsprechend kleiner.

An den linken Rand drucken wir noch eine Achsenbeschriftung, damit man die Höhe der einzelnen Balken ablesen kann. Die folgenden fünf Zeilen besorgen das für uns. Gedruckt wird der Maximalwert und die Hälfte davon. Anschließend noch elf Teilungsstriche.

In den letzten sechs Zeilen unseres Unterprogramms liegt die Schleife, die für das Zeichnen der Balken zuständig ist. Genau wie bei den Tortengrafiken wird die Schleife so oft durchlaufen, wie darzustellende Werte vorkommen. Die Zeichenfarbe wechselt innerhalb des möglichen Bereichs um eine Farbnummer pro Balken. Die Hintergrundfarbe lassen wir natürlich wieder aus.

Mit einem LINE-Befehl bringen wir die Balken auf den Bildschirm. Unterhalb der Balken wird mit LOCATE und PRINT dann schließlich noch der zugehörige Text gedruckt. Die Position richtet sich dabei in erster Linie nach der Breite der Balken. Das Ganze wiederholt sich für alle Werte, danach springt das Progamm mit RETURN zurück.

Herzlichen Glückwunsch! Sie haben auch diesen sehr mathematisch-theoretischen Teil gut überstanden. Jetzt sollten Sie erstmal ein wenig mit Ihrem neuen Utility experimentieren. Sie glauben gar nicht, was sich alles statistisch erfassen läßt. Selbst die Bundesregierung hat noch nicht alle Möglichkeiten entdeckt.

Im Laufe des Buches werden wir vor allem noch den Eingabeteil verbessern, so daß Sie zum Beispiel Werte auf Diskette abspeichern können, um Ihre Aktienkurse oder Ihre Verkaufszahlen über einen längeren Zeitraum zu beobachten. Sollten Sie statistischen und wirtschaftlichen Dingen überhaupt nichts abgewinnen können, haben wir auch noch einen Trost für Sie: Die nächsten Kapitel werden ein Programm hervorbringen, das den Künstler in Ihnen zu neuen Höchstleistungen anspornen dürfte. Und wer wäre dazu besser geeignet als Ihr Amiga?

2.8 Mehr Schein als Sein? - Menüs und Maussteuerung in BASIC

Was meinen Sie: Worin unterscheiden sich die professionellen Programme, wie Graphicraft oder Textcraft, hauptsächlich von unseren selbstgeschriebenen BASIC-Programmen? Sie sind schneller, gut. Sie sind ein Stück leistungsfähiger, auch gut. Aber einer der Hauptunterschiede ist doch: Der Anwender kann das Programm vollständig mit der Maus bedienen. Die gesamte Menüsteuerung ist durch den Mauszeiger und durch Pulldown-Menüs realisiert. "Ja," werden Sie jetzt sagen, "wenn da die Profis rangehen."

Gut, gut. Hiermit befördern wir Sie in die Gruppe der Profis und lüften damit wieder ein paar Geheimnisse der BASIC-Programmierung. Menüs, Pulldowns, Maussteuerung? Kein Problem, AmigaBASIC unterstützt diese Funktionen. Es ist gar nicht so schwer, BASIC-Programme zu schreiben, die von der Aufmachung her absolut professionell aussehen. Probieren Sie nur mal diese Zeilen. Sie finden dieses Beispiel, wie so oft, auch auf unserer Diskette. Suchen Sie nach "Menudemo".

```
MENU 1,0,1,"Auswahl"
FOR x=1 TO 9
  MENU 1,x,1,"Punkt "+CHR$(48+x)
NEXT x
FOR x=2 TO 4
```

```
  MENU x,0,0,""
NEXT x

ON MENU GOSUB Auswahl
MENU ON

Warten:
GOTO Warten

Auswahl:
LOCATE 10,20
PRINT "Sie haben Punkt";MENU(1);"gewählt.";
RETURN
```

Wenn Sie das Programm ausprobieren, glauben Sie wahrscheinlich zuerst, daß es gar nichts tut. Aber drücken Sie bitte die Menütaste der Maus (das ist die rechte Taste). Anstelle der gewohnten BASIC-Pulldowns sehen Sie nur einen einzigen Menütitel in der Kopfzeile, nämlich "Auswahl". Fahren Sie bitte mit gedrückter Maustaste auf dieses Menü und schauen Sie sich seinen Inhalt an. Zur Demonstration gibt es 9 verschiedene Menüpunkte. Wählen Sie einen davon, und das Programm druckt die Nummer auf den Bildschirm, auf die Ihre Wahl fiel.

Zugegeben, dieses Programm leistet noch nichts Großartiges, aber es zeigt Ihnen, wie einfach Menüsteuerung in AmigaBASIC programmiert werden kann, und wie professionell das Ergebnis aussieht. Um das Programm zu beenden, haben wir bisher noch keine Pulldown-Option vorgesehen. Und das "Run"-Pulldown mit seiner "Stop"-Option gibt es ja zur Zeit nicht. Drücken Sie deshalb bitte die Tastenkombination <CTRL>-<C>. Mit diesen beiden Tasten können Sie fast jedes BASIC-Programm abbrechen.

Sie werden feststellen, daß die gewohnten Pulldown-Funktionen auch nach dem Beenden des Programms nicht zurückgekehrt sind. Da wir sie aber zum Arbeiten benötigen, muß es wohl eine Möglichkeit geben, die Grundbelegung der Pulldown-Menüs wiederherzustellen. Der dazu nötige BASIC-Befehl heißt:

```
menu reset
```

Geben Sie ihn bitte im BASIC-Window ein. Danach sind die gewohnten BASIC-Pulldowns wieder verfügbar.

Kommen wir nun etwas genauer zu den einzelnen Befehlen und ihren Auswirkungen: MENU benötigt drei Zahlenwerte und den Namen, den die entsprechende Option bekommen soll. Der Befehl am Anfang unseres Beispielprogramms lautet:

```
MENU 1,0,1,"Auswahl"
```

Die erste Zahl hinter dem Befehl gibt die Nummer eines Menüs an. Bei der Standardbelegung ist das "Project"-Menü die Nummer 1, "Edit" die Nummer 2, "Run" die Nummer 3 und "Windows" die Nummer 4. Ingesamt können in AmigaBASIC 10 verschiedene Pulldown-Menüs definiert werden.

Die zweite Zahl bestimmt den Menüpunkt innerhalb des angegebenen Menüs. Nummer 0 ist der Titel des Menüs, die Nummern 1 bis (höchstens) 19 geben die verschiedenen Optionen an.

An dritter Stelle kann nur eine von drei Zahlen stehen.

0 schaltet den angegebenen Menüpunkt aus. Der Punkt ist dann nicht mehr verfügbar, er wird in der etwas schwer lesbaren Schrift gedruckt, die Sie ja schon kennen. Diese Schrift nennt man auch "Geisterschrift". Ist die zweite Zahl hinter dem MENU-Befehl eine 0, bezieht sich das Aus- oder Anschalten auf das ganze Pulldown.

1 bewirkt das Gegenteil von 0. Die angegebene Menü-Option bzw. das gesamte Pulldown wird angeschaltet, steht wieder zur Auswahl.

2 schließlich wirkt wie 1, mit einem Unterschied: Der angegebene Menüpunkt wird mit einem Häkchen versehen. (Es ist der Typ Häkchen, der auch in Schule und Beruf immer das beruhigende Gefühl vermittelt, sich für das Richtige entschieden

zu haben. Im Gegensatz zu Kreuzchen, die man bei Wahlen macht.) Mit dieser Hilfe können Sie dem Benutzer Ihres Programms leicht anzeigen, welcher Modus zur Zeit aktiv ist.

Für den Titel des jeweiligen Menüpunkts geben Sie einen beliebigen Stringausdruck an, entweder den Text in Anführungszeichen oder eine Stringvariable, die den Text beinhaltet. Wenn Sie den String weglassen, verändert der MENU-Befehl lediglich die Parameter einer bereits vorhandenen Pulldown-Option (meistens werden Sie wohl die dritte Zahl verändern). Sie können auch, wie in unserem Beispielprogramm, mit dem CHR$-Befehl einzelne Zeichen angeben oder zum Text hinzufügen. Wenn Sie sich fragen, warum wir zu den Zahlen in 'x' ausgerechnet den Wert 48 addieren, dann schauen Sie sich bitte die ASCII-Tabelle an, die Sie beim CHR$-Befehl im Anhang B finden.

```
menu 1,0,1,"Test"
```

tauft das Pulldown Nummer 1 auf den Namen "Test". Geben Sie gleich den nächsten Befehl ein. Lassen Sie dabei bitte vor dem Wort "Nummer" zwei Leerstellen Platz.

```
menu 1,1,1,"  Nummer 1"
```

Jetzt dürfen Sie sich beruhigt das erste Pulldown anschauen. Mit dem zweiten Befehl haben wir dort einen eigenen Unterpunkt definiert. Allerdings kann AmigaBASIC verständlicherweise im Direktmodus (das heißt also ohne Programmkontrolle) mit dieser Option nichts anfangen, also nicht darauf reagieren. Wenn Sie das gesamte Menü Nummer 1 abschalten wollen, verwenden Sie:

```
menu 1,0,0
```

Um dies rückgängig zu machen, verwenden Sie den Befehl:

```
menu 1,0,1
```

Der nächste Befehl versieht den ersten Punkt des ersten Menüs mit einem Häkchen, zur Kennzeichnung dieser Menüoption:

```
menu 1,1,2
```

Wie Sie in unserem Beispielprogramm sehen, ist es nötig, die voreingestellten BASIC-Pulldowns einzeln zu löschen, falls im Programm weniger als vier eigene Menüs definiert werden. Das Löschen geht so:

```
for x=1 to 4 : menu x,0,0,"" : next x
```

Nach dieser Zeile gibt es überhaupt keine Pulldowns mehr. Diese Möglichkeit kann in manchen Programmen auch recht sinnvoll sein. Sollten Sie schon bald große Sehnsucht nach den gewohnten und geschätzten Menüs verspüren, wissen Sie ja noch, wie Sie sie zurückbekommen:

```
menu reset
```

Mit dem MENU-Befehl können Sie in Ihren Programmen zunächst alle Menüs nach Ihren Vorstellungen anrichten, oh Verzeihung - wir meinen natürlich: einrichten.

Bisher haben wir die Überwachung und Auswertung der Menüs noch nicht ins Spiel gebracht. Die Methode, nach der diese Überwachung funktioniert, gehört zu den faszinierendsten Möglichkeiten von AmigaBASIC: Wenn Sie sich das Beispielprogramm noch einmal genau ansehen (oder es mit TRACE verfolgen), werden Sie feststellen, daß das Hauptprogramm eigentlich nur aus der Schleife 'Warten:' besteht. Diese Schleife springt ständig auf sich selbst. Das ist auch wieder so etwas, was nur Computer fertigbringen, ohne dabei verrückt zu werden.

Alles andere funktioniert nach dem Prinzip des "Event Trapping" (deutsch etwa: Ereignisüberwachung). In dem Augenblick, wo ein bestimmtes Ereignis eintritt, das von AmigaBASIC überwacht werden soll, wird das gegenwärtige Programm unterbrochen und ein vorher festgelegtes Unterprogramm aufgerufen. Ist dieses Unterprogramm zu Ende, macht AmigaBASIC genau an der Stelle weiter, wo es vorher unterbrochen hat. Diese Art zu programmieren, ist nicht nur überaus praktisch, sondern auch sehr effektiv. Sie können verschiedene Unterroutinen schreiben,

die sich um Programmabbruch, Kollision von Grafikobjekten, Fehlermeldungen, Menü- und Maus-Steuerung oder Timer-Abläufe (also zeitlich festgelegte Unterbrechungen) kümmern, während das Hauptprogramm etwas völlig anderes tut.

Der Befehl, mit dem Sie AmigaBASIC mitteilen, welches Unterprogramm angesprungen soll, wenn der Benutzer eine Menü-Auswahl vornimmt, lautet:

ON MENU GOSUB ...

Dahinter können Sie ein Label oder eine Zeilennummer angeben. Jede Möglichkeit des Event Trapping hat noch eine eigene "Sicherung" und muß erst ausdrücklich "scharf" gemacht, bzw. aktiviert werden. Dazu verwenden Sie für die Menü-Überwachung den Befehl

MENU ON

Wollen Sie das Menü-Trapping vorübergehend unterbrechen, verwenden Sie MENU STOP. Zum endgültigen Abschalten gibt es MENU OFF. Der Unterschied zwischen diesen beiden Befehlen liegt in der "Wahrnehmung" des Ereignisses.

Bei MENU STOP registriert der Amiga die Menüauswahl, reagiert aber nicht sofort darauf. Er wartet, bis Event Trapping wieder zugelassen wird. Nach dem nächsten MENU ON werden dann auch die Ereignisse ausgewertet, die während der Unterbrechung stattfanden. Dagegen unterdrückt MENU OFF die Überwachung vollständig. Der Amiga reagiert nicht auf die Menüauswahl, bis das Event Trapping durch MENU ON wieder zugelassen wird.

Die Arbeitsweise von ON, OFF und STOP ist identisch für alle anderen möglichen Ereignisse. Wir werden von Fall zu Fall darauf zurückkommen. Da können Sie dann auch am praktischen Beispiel die Unterschiede zwischen OFF und STOP besser erkennen.

Die letzte Frage, die wir jetzt noch beantworten sollten, lautet: "Was kann ich das von ON MENU GOSUB aufgerufene Unterprogramm alles machen lassen?" Wie gesagt, wird bei jeder Anwahl einer Menüoption durch den Benutzer das angegebene Unterprogramm aufgerufen. Dieses Unterprogramm muß nun zuerst mal herausfinden, welcher Menüpunkt überhaupt gewählt wurde. Dazu gibt es zwei Funktionen, nämlich MENU(0) und MENU(1). MENU(0) liefert die Nummer des gewählten Pulldowns, diese Zahl entspricht der ersten Angabe beim MENU-Befehl. MENU(1) liefert die Nummer der gewählten Option innerhalb des gewählten Pulldowns, diese Zahl entspricht der zweiten Angabe im MENU-Befehl. Damit erhält Ihr BASIC-Programm alle Informationen, die es braucht, um auf die Menü-Auswahl zu reagieren.

Das folgende Beispiel soll Ihnen das verdeutlichen: Wir erweitern das Beispielprogramm um einen Menüpunkt. Fügen Sie den Befehl bitte genau zwischen die beiden FOR...NEXT-Schleifen im ersten Teil des Programms ein:

```
menu 1,10,1,"Beenden"
```

Das Unterprogramm 'Auswahl:' verändern Sie nun so, daß dabei folgendes herauskommt:

```
Auswahl:
IF MENU(1)=10 THEN MENU RESET : END
LOCATE 10,20
PRINT "Sie haben Punkt";MENU(1);"gewählt."
RETURN
```

Die Version, die nun entstanden ist, finden Sie als "Menudemo2" auch auf der Diskette im Buch.

Und so funktioniert's: Das 'Auswahl:'-Programm überprüft zuerst, ob der gewählte Punkt die Nummer 10 hat. Nummer 10 bedeutet "Beenden". (Wer jetzt an Bo Dereks Film "10" gedacht hat, ist vielleicht ein bißchen enttäuscht...) Trifft das zu, wird

mit MENU RESET die Menü-Grundbelegung wiederhergestellt, und das Programm bricht mit END ab. In allen anderen Fällen verhält sich das Programm so wie bisher.

Da es in unserem Beispiel nur ein Pulldown gibt, brauchen wir uns um die Nummer des Pulldowns, also den Wert MENU(0), nicht weiter zu kümmern. In größeren Programmen mit mehreren Pulldowns müssen Sie erst mit MENU(0) das gewählte Menü feststellen, bevor Sie mit MENU(1) die einzelnen Punkte abprüfen können.

Ist soweit alles klar? Sollte es Ihnen noch ein wenig an umfangreicheren Anwendungsbeispielen mangeln: Es dauert gar nicht mehr lange, bis wir Ihnen anhand eines großen und leistungsfähigen BASIC-Utilities die Menüsteuerung und manches andere vorführen können. Vorher fehlt uns aber noch eine andere wichtige Grundlage: Die Abfrage und Programmierung der Maus. Zur Demonstration bieten wir Ihnen das wohl kürzeste Malprogramm auf dem Amiga:

```
ON MOUSE GOSUB Zeichnen
MOUSE ON

Warten:
GOTO Warten

Zeichnen:
  WHILE MOUSE(0)<>0
    PSET (MOUSE(1),MOUSE(2))
  WEND
RETURN
```

Bedenkt man die geringe Arbeit, die Sie mit der Eingabe hatten, ist das Programm doch schon ganz ordentlich, oder nicht? Und wenn Sie sich die Arbeit noch weiter erleichtern wollen, können Sie das Ganze auch als "Mausdemo" von der beiliegenden Diskette laden.

Sie sehen jedenfalls, daß Maus-Abfragen vom Grundprinzip her genauso funktionieren wie Menü-Abfragen. Mit dem Befehl

ON MOUSE GOSUB ...

geben Sie das Unterprogramm an das sich bei Bedarf um die Maus kümmern soll.

```
MOUSE ON
```

aktiviert das Ganze, und die folgende Endlosschleife vertritt ein laufendes Programm.

Nur im Programmteil 'Zeichnen:' gibt es einiges Neue und Erklärenswerte: Zum Beispiel den WHILE...WEND-Befehl. Diese beiden Befehle bieten eine neue Möglichkeit, Schleifen zu programmieren. "While" heißt auf Deutsch "Während, solange" und "wend" ist wahrscheinlich ein Kunstwort aus "WHILE End", es könnte aber auch übersetzt "wenden, sich umdrehen" bedeuten. Sie sehen: So ganz einfach sind die Überlegungen der BASIC-Sprachschöpfer auch nicht immer nachzuvollziehen.

Diese Schleifenstruktur dient dazu, einen Programmteil solange ausführen zu lassen, wie ein bestimmter Zustand anhält. Eine einfache Schleife, die auf einen Tastendruck wartet, sieht so aus:

```
WHILE INKEY$="" : WEND
```

Dieses Beispiel können Sie sogar im BASIC-Window ausprobieren. WHILE...WEND ist oft einfacher und eleganter als FOR...NEXT oder IF...THEN, obwohl man mit ein wenig Aufwand jede der drei Schleifenarten durch eine der beiden anderen ersetzen kann. Die Bedingung, die die WHILE...WEND-Schleife im Beispielprogramm abfragt, heißt:

```
MOUSE(0)<>0
```

Ihnen wird aufgefallen sein, daß Sie nur zeichnen können, während die linke Maustaste gedrückt ist. Die Funktion MOUSE(0) gibt Auskunft über den Zustand der Maustaste. Sie

hat den Wert 0, wenn keine Maustaste gedrückt ist. Falls sie doch gedrückt ist, gibt es verschiedene Möglichkeiten, die wir aber an dieser Stelle noch nicht alle besprechen wollen. Wenn wir beim Programmieren einer bestimmten Variante über den Weg laufen, werden wir sie dort erklären. Wer sofort alles wissen will, kann aber auch gern gleich unter MOUSE(x) in der Befehlsbeschreibung im Referenzteil dieses Buches nachschauen. Nicht nur MOUSE(0), sondern auch MOUSE(1) und MOUSE(2) kommen in unserem Beispiel vor. Die beiden sind fast noch einfacher als MOUSE(0):

MOUSE(1) liefert die x-Koordinate der Maus.
MOUSE(2) liefert die y-Koordinate der Maus.

Beide Werte werden in Pixels angegeben, wobei sich AmigaBASIC beim Zählen automatisch an den aktuellen Screen anpaßt. Das macht das Zeichnen mit der Maus so einfach.

Vielleicht haben Sie jetzt schon einen Verdacht, welche Art von Programm unser vielgepriesenes Utility sein könnte. Was könnte wohl am besten dazu geeignet sein, alle denkbaren Funktionen von Maus und Pulldowns zu demonstrieren, dabei auch noch ein nützliches Hilfsmittel ergeben und vor allem auf dem Amiga interessant sein?

Wir werden im nächsten Kapitel ein eigenes Malprogramm schreiben. Ein Malprogramm, das fast alle Grafikfähigkeiten von AmigaBASIC unterstützt, einfach zu bedienen ist und dabei wirklich tolle Bilder erzeugen kann. Was es außerdem noch bietet? Ein paar besondere Bonbons - aber lesen Sie selbst...

2.9 Das große Tippen - das AmigaBASIC-Malprogramm

Wir wollen Ihnen jetzt ein Utility vorstellen, das Ihnen hilft, in BASIC Bilder und Zeichnungen zu erzeugen. Das Ding zu entwickeln, war wirklich ein gutes Stück Arbeit. Wir sagen das jetzt nicht, um uns zu loben, sondern um Ihnen klar zu machen, warum es so lang geworden ist. Sie sollten also etwas Zeit mitbringen, wenn Sie sich an das bisher größte Projekt in diesem

Buch wagen. (Vielleicht nehmen Sie auch noch gleich ein paar Nüßchen, Chips und was zu Trinken mit...) Sind Sie soweit? Verabschieden Sie sich von Ihrer Familie, schließen Sie die Tür hinter sich: Es geht los. Aber es wird sich lohnen. Sowohl vom Ergebnis als auch vom Lernerfolg her.

Wenn Sie bisher alle Beispiele dieses Buches mitgemacht haben, besitzen Sie von uns schon zwei größere Hilfsprogramme: Den Videotitelgenerator, mit dem Sie aus Texten und bewegten Grafikobjekten einen Video-Vorspann erzeugen können und das Balken-/Tortengrafik-Utility, das statistische Daten in Grafiken umsetzt.

Bei der Entwicklung unser Beispiele verfolgten wir vor allem ein Ziel: Wir wollen Ihnen Programme in die Hand geben, die Ihnen auch nach der Lern-Phase einen möglichst hohen Nutzen bieten. Gerade auf dem Gebiet der Grafik gibt es einen wichtigen Punkt, um den Anwendungswert einzelner Programme zu erhöhen: Die Programme sollten untereinander *kompatibel* sein. Dadurch wird es für Sie zum Beispiel möglich, mit dem Malprogramm Bilder zu erzeugen, die Sie später im Videotitel-Programm einlesen und als Hintergründe verwenden können. Oder auch die Bilder der "großen" Malprogramme, wie Graphicraft oder DeluxePaint, einzulesen und in AmigaBASIC weiter zu bearbeiten. Ja, das geht! Wir haben doch gesagt, es wird sich lohnen. Und wir versuchen, immer zu halten, was wir versprechen.

Ein wichtiges Glied in dieser Kompatibilitätskette ist unser Malprogramm. Wir glauben, daß mit ihm ein sehr hilfreiches und leistungsfähiges Utility entstanden ist. Viele der Möglichkeiten, die Sie von kommerziellen Grafik-Programmen kennen, werden Sie auch in unserem BASIC-Programm finden. Natürlich kann ein Programm, das so viel bietet, nicht so kurz sein wie die Beispiele, mit denen wir nur einzelne Befehle erläutern. Im Klartext heißt das, daß im Lauf der nächsten beiden Kapitel insgesamt 9 Seiten Listing auf Sie warten, die in den Amiga eingegeben werden müssen. Natürlich haben Sie auch hier die Möglichkeit, das ganze Programm von unserer beigelegten Dis-

kette zu laden. Allerdings sollten Sie sich trotzdem Stück für Stück mit den Beschreibungen in den folgenden Kapitel auseinandersetzen und sie mit dem Programm vergleichen.

Wie gesagt: Für diese Kapitel werden Sie einige Zeit brauchen. Nehmen Sie sich diese Zeit, niemand drängt Sie. Machen Sie auch zwischendrin ruhig einige Pausen - wir sind nicht sauer, wenn Sie uns mal ein bißchen in den Bücherschrank stellen.

Natürlich erklären wir das Programm ausführlich und weisen dabei auf besondere Befehle, Optionen oder spezielle Tricks hin. Und damit kann's eigentlich losgehen. Wenn Sie sich fürs Tippen entschieden haben, dann geben Sie bitte nacheinander die Programmteile aus diesem Kapitel ein. Übernehmen Sie dabei möglichst die strukturierte Schreibweise. Das Programm wird dadurch nicht nur übersichtlicher, sondern Sie werden auch Eingabefehler leichter finden und ausmerzen können.

```
Vorbereitungen:
 Farben=5 : Maxfarbe=2^Farben-1
 DIM Zeiger(4,1),Altfarbe(4),Farben%(31,2)
 DIM Muster%(7),Allemuster%(8,7),Voll%(1)
 Zart=1 : Zfarbe=1 : Ffarbe=2 : Modus=1

 Voll%(0)=&HFFFF : Voll%(1)=&HFFFF
 FOR x=0 TO 7
   Muster%(x)=&HFFFF
   Allemuster%(0,x)=&HFFFF
 NEXT x
 FOR x=1 TO 8
   FOR y=0 TO 7
     READ Allemuster%(x,y)
   NEXT y
 NEXT x

 DATA 24672,1542,24672,1542,24672,1542,24672,1542
 DATA -13108,13107,-13108,13107,-13108,13107,-13108,13107
 DATA 26214,13107,-26215,-13108,26214,13107,-26215,-13108
 DATA -13108,-26215,13107,26214,-13108,-26215,13107,26214
```

```
    DATA -258,-258,-258,0,-4113,-4113,-4113,0
    DATA -8185,-8197,-18019,-20491,-20467,-8197,-8185,-1
    DATA 0,0,1632,4080,4080,2016,384,0
    DATA 960,1984,3520,6592,16320,25024,-3104,0

  SCREEN 1,320,200,Farben,1
  WINDOW 2,"AmigaBASIC Zeichenprogramm",,16,1
  WINDOW CLOSE 3
  WINDOW CLOSE 4
  WINDOW 2
```

Das Programm beginnt mit einem Vorbereitungsteil. Hier werden alle Variablen auf ihre Grundwerte gesetzt, die Datenfelder mit ihrer Ausgangsbelegung versehen und Screen und Window für die Bildschirmausgabe angelegt.

In der Variablen 'Farben' legen Sie wie immer die Anzahl der verwendeten Bit-Ebenen fest. Diesmal stehen die Zahlen 3 (8 Farben), 4 (16 Farben) oder 5 (32 Farben) zur Auswahl. Im Normalfall werden Sie mit 32 Farben arbeiten, denn unser Programm arbeitet im niedrigauflösenden Modus. Auf einem Amiga mit mindestens 512KByte RAM dürfte es mit dem Speicherplatz also keine Probleme geben.

Beim Feld 'Farben%' fällt Ihnen sicher zuerst die ungewohnte Schreibweise mit dem Prozentzeichen hinter dem Namen auf. AmigaBASIC bietet verschiedene Variablentypen. Zwei kennen Sie bisher schon, nämlich die normalen, numerischen Variablen ('a', 'b', 'c', 'Hallo'...) und die Stringvariablen ('a$', 'b$', 'c$', 'Hallo$'). Stringvariablen werden durch ein $-Zeichen hinter dem Variablennamen gekennzeichnet. Eine vergleichbare Funktion hat auch das %-Zeichen. Damit werden Integer-Variablen gekennzeichnet.

Sie kennen ja schon die BASIC-Funktionen INT und CINT. Daher wissen Sie, daß Integer-Zahlen immer ganze Zahlen sind, also keinen Nachkommateil haben. Es gibt die Möglichkeit, Variablen oder Felder von vornherein als Integer-Zahlen zu deklarieren. Diese Zahlen können nie einen Nachkommateil haben

und benötigen daher auch weniger Platz im Speicher. Der Grund dafür ist die Art, wie Zahlen vom Amiga intern verwaltet werden. Aus denselben Gründen können Integer-Zahlen nur Werte zwischen -32768 und 32767 annehmen. Im Zwischenspiel 4 werden wir darauf genauer eingehen. Und im nächsten Teil dieses Buches werden wir noch verschiedene andere Variablentypen kennenlernen. Probieren Sie im BASIC-Window:

```
a=7/3 : a%=7/3 : ? a,a%
```

Sie sehen, daß die Variable 'a%' keine Stellen hinter dem Komma annimmt. Da im Malprogramm verschiedene Daten, wie Farben und Muster, nur ganze Zahlen sein können, verwenden wir dafür Integer-Felder. Das Feld 'Farben%' hat die Dimensionen (31,2), kann also 32 * 3 Elemente aufnehmen. (Erinnern Sie sich noch an mehrdimensionale Felder? Stellen Sie sich solche Felder am besten als Tabelle vor, in der in an einer bestimmten Position immer eine bestimmte Zahl steht.) Wir können im Programm bis zu 32 verschiedene Farben verwenden, wobei pro Farbe drei Werte zu merken sind (R, G und B).

Im Vorbereitungsteil definieren wir unter anderem auch Füllmuster. Sie können in AmigaBASIC Muster angeben, mit denen Flächen ausgefüllt werden. Wie das genau geht, zeigen wir im nächsten Kapitel. Bisher müssen Sie zu diesem Thema nur eines wissen: Man benötigt dazu Integerfelder, in denen das Aussehen der Muster gespeichert wird. Wir machen das in unserem Programm in den Feldern 'Muster%', 'Allemuster%' und 'Voll%'.

Im Feld 'Muster%' wird das Aussehen des aktuellen Musters festgelegt, im Feld 'Allemuster%' befinden sich (wie der Name schon sagt) alle Muster. Unser Malprogramm bietet nämlich die Möglichkeit, zum Füllen von Flächen immer eines aus 9 verschiedenen Mustern auszuwählen. Die Definitionen aller 9 Muster stehen in 'Allemuster%'. Und in 'Voll%' ist ein ganz spezielles Muster abgelegt, das volle, einfarbige Flächen produziert. Die brauchen wir nämlich auch von Zeit zu Zeit.

Im Vorbereitungsteil werden die Felder mit Grundwerten gefüllt. Zu den Befehlen, die dabei vorkommen, zwei Erklärungen: Da gibt es schon wieder eine spezielle Schreibweise für Zahlen. Was hat das auf den ersten Blick kurios aussehende &HFFFF zu bedeuten? Nun, dabei handelt es sich um eine *Hexadezimal-*Zahl. Das ist eine für Programmierer besonders geeignete Schreibweise für Zahlen. Alles weitere über diese Zahlen finden Sie im Zwischenspiel 4. Mit Hexadezimal-Zahlen kann AmigaBASIC auch arbeiten. Der Wert &HFFFF bewirkt bei der Musterdefinition, daß alle Punkte im Raster angeschaltet sind. So entsteht eine volle, gleichförmige, einfarbige Fläche. Dieses volle Muster legen wir zu Beginn des Programms ab in den Feldern 'Voll%' und 'Muster%' und im ersten der neun Auswahlmuster.

Wir wollen Ihnen gleich nach dem Starten des Programms eine Auswahl an vorhandenen Mustern bieten. Deshalb haben wir 9 Stück für Sie vorbereitet. Die nötigen Werte werden mit READ und DATA eingelesen. Moment bitte, womit?

Die Befehle READ und DATA sind ein Gespann, mit dessen Hilfe es möglich ist, Daten innerhalb eines Programms abzulegen, dann einzulesen und zu verwenden. (Deshalb auch READ = lesen und DATA = Daten). Mit Daten meinen wir zum Beispiel Zahlen oder Worte, aber auch jede beliebige Mischform aus Zahlen, Zeichen und Buchstaben. Die Daten stehen in einer Programmzeile, die mit dem Befehl DATA beginnt. Dahinter stehen, durch Kommas getrennt, die einzelnen Werte. Mit dem READ-Befehl lesen Sie diese Daten in Variablen ein. Das READ/DATA-Gespann ist nur innerhalb eines Programms sinnvoll. Daher können wir Ihnen kein Beispiel im Direktmodus zeigen. Die folgenden fünf Zeilen gehören nicht zum Malprogramm. Wenn Sie das Beispiel ausprobieren wollen, vergessen Sie bitte auf keinen Fall, vorher Ihr Malprogramm abzuspeichern!

```
READ a$,b$,c$,d
PRINT "Ich habe mir einen ";a$;" gekauft."
PRINT "Das ist ein toller ";b$;", der mir viel ";c$;" macht."
```

```
PRINT "Mit ihm kann man ";d;" Dinge machen."
DATA Amiga,Computer,Spass,1000
```

Sehen Sie, wie's funktioniert? Den verschiedenen Variablen hinter READ wird immer der nächste, noch nicht gelesene Wert aus der aktuellen DATA-Zeile zugewiesen. In einem Programm fängt das Lesen mit READ immer in der ersten DATA-Zeile an, egal wo sie steht. Dann kommt ein Wert nach dem anderen an die Reihe. Sie müssen darauf achten, daß die DATAs und die Variablentypen zusammenpassen. Wenn der nächste DATA-Wert Buchstaben enthält und Sie ihn z.B. in eine Zahlenvariable einlesen wollen, erhalten Sie einen "Type mismatch"-Error ("Datentyp durcheinander gebracht"). Wenn alle DATAs gelesen sind und wieder ein READ-Befehl im Programm ausgeführt werden soll, meldet AmigaBASIC einen "Out of DATA"-Error.

Der erste Block von DATA-Befehlen im Programm hat das Aussehen der neun Grund-Muster gespeichert. Diese Werte werden ins Feld 'Allemuster%' eingelesen. Dann definieren wir den Ausgabe-Screen und das Ausgabe-Window.

Wie wir schon einmal erwähnt haben, läuft unser Malprogramm in der niedrigsten Auflösungsstufe (320 * 200 Punkte). Das hat verschiedene Gründe. Der Hauptgrund: Wir wollen später ja fertige Bilder bearbeiten, die z.B. mit dem Programm Graphicraft erzeugt wurden. Fast alle professionellen Malprogramme verwenden die niedrige Auflösung. Sie reicht völlig aus, um überzeugende Bilder darzustellen, und sie ermöglicht 32 verschiedene Farben. Wer einige Beispielbilder auf dem Amiga gesehen hat, kann bestätigen: Die Vielfalt an Farbtönen macht die relativ grobe Auflösung wett.

Vielleicht fragen Sie sich, warum wir für die Bildschirmdarstellungen in unseren Beispielprogrammen den PAL-Bereich des Bildschirms nicht nutzen. Auch hier ist der Grund, daß wir zu den üblichen Grafikprogrammen wie Graficraft oder DPaint kompatibel bleiben wollen. Und alle guten Grafikprogramme kommen zur Zeit nun mal aus Amerika und nutzen deshalb den PAL-Bereich nicht.

Die nächsten beiden Programmzeilen lassen die Windows 3 und 4 verschwinden, falls sie sich schon auf dem Bildschirm befinden (was passieren könnte, wenn das Programm abgebrochen und neu gestartet wird).

Geben Sie jetzt bitte den nächsten Teil des Malprogramms ein:

```
FOR x=0 TO 31
  READ r,g,b
  PALETTE x,r/16,g/16,b/16
  Farben%(x,0)=r : Farben%(x,1)=g : Farben%(x,2)=b
NEXT x

DATA 0,0,3, 15,15,15, 0,3,12, 15,0,0
DATA 0,14,15, 15,0,15 ,3,10,1, 15,14,0
DATA 15,8,0, 10,0,14, 8,5,0, 11,8,3
DATA 2,11,0, 15,10,15, 0,0,9, 7,15,0
DATA 14,12,0, 15,2,3, 0,0,0, 15,11,10
DATA 0,6,8, 3,3,3, 4,4,4, 5,5,5
DATA 6,6,6, 7,7,7, 8,8,8, 9,9,9
DATA 11,11,11, 13,13,13, 0,0,15, 12,15,12
```

Das Feld 'Farben%' wird in diesem Teil mit seinen Grundwerten geladen. Im Lauf des Programms stehen in 'Farben%' die R-, G- und B-Anteile der verwendeten Farben als Zahlen zwischen 0 und 15. Am Anfang tritt aber ein Problem auf: Sie wissen vielleicht noch vom Videotitel-Programm, daß es keinen BASIC-Befehl gibt, um die aktuellen Werte der Farbregister auszulesen. Sie können zwar mit PALETTE geändert werden, aber wir kennen keine Möglichkeit, die aktuellen Werte von BASIC aus festzustellen.

Wenn die Farben im Programm zum ersten Mal geändert werden sollen, würde ihre Grundeinstellung zunächst auf Schwarz fallen (R=0, G=0, B=0). Das ergäbe auf dem Bildschirm einen so trostlosen Eindruck, daß wir Ihnen so etwas nicht zumuten konnten. Früher oder später würde ja auch der Nacht-und-Nebel-Effekt eintreten, denn schwarze Zeichnungen auf schwarzem Grund sind nun mal extrem schwer zu erkennen.

Wir haben deshalb die RGB-Werte von 32 Farben in DATA-Zeilen abgelegt, die zu Beginn in das Feld 'Farben%' eingelesen werden und dort solange bleiben, bis sie vom Programm geändert werden.

```
Pulldown:
  MENU 3,0,0,""
  MENU 4,0,0,""
  MENU 1,0,1,"Programm"
  MENU 1,1,1,"Zeichnen"
  MENU 1,2,1,"Farben ändern"
  MENU 1,3,1,"Muster ändern"
  MENU 1,4,1,"Bild laden"
  MENU 1,5,1,"Bild speichern"
  MENU 1,6,1,"Bild löschen"
  MENU 1,7,1,"Ende"
  MENU 2,0,1,"Zeichenart"
  MENU 2,1,2,"  Freihand dünn"
  MENU 2,2,1,"  Freihand dick"
  MENU 2,3,1,"  Punkte"
  MENU 2,4,1,"  Spraydose"
  MENU 2,5,1,"  Linien"
  MENU 2,6,1,"  Rechtecke"
  MENU 2,7,1,"  Blocks"
  MENU 2,8,1,"  Vielecke"
  MENU 2,9,1,"  Kreise"
  MENU 2,10,1,"  Ausmalen"
  MENU 2,11,1,"  Radiergummi"
  MENU 2,12,1,"  Text"
```

Was hier passiert, ist schon durch den ständig auftauchenden MENU-Befehl unübersehbar: Die Pulldown-Menüs werden eingerichtet. Zuerst löschen wir die Menüs 3 und 4, denn in unserem Programm kommen wir mit zwei Pulldowns aus. Im Menü 1 liegen die Optionen zur Programmsteuerung. Damit werden die einzelnen Funktionen ausgewählt. Im Menü 2 finden Sie die Werkzeuge, die Ihnen beim Zeichnen zur Verfügung stehen. Die gerade aktive Zeichenart wird ständig durch ein Häkchen gekennzeichnet. Deshalb auch bitte beim Eintippen vor den einzelnen Menüpunkten zwei Stellen freilassen.

Schon sind alle nötigen Vorbereitungen durchgeführt. Nun schließt sich nahtlos der eigentliche Kern des Programms an, die 'Hauptschleife:'.

```
Hauptschleife:
  ON MENU GOSUB Menuauswahl
  ON MOUSE GOSUB Maus
  MENU ON
  MOUSE ON

  WHILE -1
  WEND
```

Das Event Trapping für Menüs und Maussteuerung wird aktiviert und mit zuständigen Unterprogrammen versehen. Dahinter sehen Sie eine WHILE...WEND-Schleife. Auch wenn Sie's nicht glauben: Es fehlt nichts, es handelt sich auch nicht um einen Druckfehler. Die Zahl -1 hinter WHILE stellt für AmigaBASIC eine logisch "wahre", d.h. zutreffende Bedingung dar. Wir werden über dieses Thema noch genauer im nächsten Zwischenspiel sprechen. Jedenfalls bewirken die beiden Zeilen eine leere Endlosschleife.

Solange der Anwender die Maus nicht benutzt (also weder ein Menü anwählt, noch die Maus bewegt oder mit einer Maustaste klickt), läuft das Programm nur in dieser Schleife. Sämtliche Funktionen unseres Malprogramms werden durch Event Trapping gesteuert. Falls Sie schon einmal auf einem anderen Computer programmiert haben, wird Ihnen diese Methode ziemlich ungewöhnlich vorkommen, doch sie ermöglicht sehr flexible und leistungsfähige Programme. Natürlich lassen sich Event Trapping und "normale" Programmierung auch miteinander verbinden. Wie Sie ein Programm anlegen, hängt von der Aufgabenstellung ab.

Im Anschluß an das Hauptprogramm folgen die Programmteile, die für die Menüsteuerung gebraucht werden:

```
Menuauswahl:
   Men=MENU(0)
   Menpunkt=MENU(1)
   ON Men GOTO Projekt, Zeichenart

Maus:
   IF Modus=1 THEN ON Zart GOSUB DueFreihand,DiFreihand,Punkte,Spraydose,
Linien,Rechtecke,Block,Flaechen,Kreis,Fuellen,Radiergummi,Text
   IF Modus=2 THEN GOSUB Farbeinstellung : IF EndOK=1 THEN GOSUB Farbaus
   IF Modus=3 THEN GOSUB Musterdef : IF EndOK=2 THEN GOSUB Musteraus
   IF Modus=4 THEN GOSUB RGBDef : IF EndOK=3 THEN Modus=2 : GOSUB Farbaus
wahl
RETURN

Projekt:
   IF Menpunkt=1 THEN GOSUB Farbaus : GOSUB Musteraus
   IF Menpunkt=2 THEN GOSUB Musteraus : MENU 2,0,0 : Modus=2 : GOSUB Farb
auswahl
   IF Menpunkt=3 THEN GOSUB Farbaus : MENU 2,0,0 : Modus=3 : GOSUB Muster
editor
   IF Menpunkt=6 AND Modus=1 THEN OK=0 : GOSUB Abfrage : IF OK=1 THEN Ade
f=0 : AREAFILL : CLS
   IF Menpunkt=7 THEN GOSUB Farbaus : GOSUB Musteraus : OK=0 : GOSUB Abfr
age : IF OK=1 THEN Ende
RETURN

Zeichenart:
  MENU 2,Zart,1
  Zart = MENU (1)
  MENU 2,Zart,2
RETURN
```

'Menuauswahl:' fragt die angewählte Menü-Option ab. In der Variablen 'Men' steht das Menü und in 'Menpunkt' steht - na was wohl - der Menüpunkt eben. Je nachdem, welches Menü gewählt wurde, wird 'Projekt:' oder 'Zeichenart:' angesprungen. Hier gibt es eine neue Variante des GOTO-Befehls kennenzulernen.

ON...GOTO verzweigt zu einem betimmten Label bzw. einer Zeilennummer, abhängig vom Wert einer Variablen. Gönnen wir uns ein kulinarisches Beispiel, wo es ja im weiteren Sinne auch um ein Menü geht. Die folgende Zeile tippen Sie bitte nicht ab, sie gehört nicht zum Malprogramm:

```
ON x GOTO Aperitif,Vorspeise,Suppe,Hauptgericht,Käse,Nachspeise
```

Hat x den Wert 1, springt AmigaBASIC zum Label Aperitif. Bei x=2 wird Vorspeise aufgerufen. Bei x=3 ist die Suppe dran. Die Bedingung x=4 führt zum Hauptgericht, x=5 schließt den Magen mittels Käse und x=6 serviert die Nachspeise.

Ist x kleiner als 1 oder größer als 6, wird der ON...GOTO-Befehl überhaupt nicht ausgeführt und das Programm macht bei der nächsten Zeile weiter. Wer sich nicht im üblichen Rahmen bewegen kann, bekommt eben gar nichts zu essen. Wenn Sie jetzt Hunger bekommen haben sollten, beschweren Sie sich nicht bei uns: Wir haben extra gesagt, Sie sollten noch was zum Knabbern mitnehmen...

Am Anfang des Programmteils 'Maus:' finden Sie eine ON...GOSUB-Zeile, die auch nicht gerade klein und bescheiden ist. ON...GOSUB funktioniert übrigens genauso wie ON...GOTO. Nur kehrt das Programm beim nächsten RETURN an die Stelle zurück, wo der GOSUB-Aufruf steht. Aber immer schön der Reihe nach: 'Maus:' ist für die Kontrolle und Auswertung der Maus zuständig. Wenn der Benutzer des Programms mit der linken Maustaste klickt, springt AmigaBASIC zunächst zu diesem Label. Das Klicken mit der Maus kann ja sehr unterschiedliche Absichten bekunden.

Was der Anwender mit dem Mausklick bezweckt, hängt davon ab, in welchem Modus sich das Programm gerade befindet. Um das festzustellen, gibt es die Variable 'Modus'. Die verschiedenen Unterprogramme versorgen sie mit dem richtigen Wert. Hat 'Modus' den Wert 1, wird gerade gezeichnet. Dieser Modus wird im Programm durch die erste Option des ersten Pulldowns

gewählt. Die Variable 'Zart' hat nichts mit zart oder zartbitter zu tun. Warum denken wir eigentlich ständig ans Essen? Wird Zeit, daß wir mal wieder zum Luigi gehen...

'Zart' ist eine Abkürzung für "Zeichenart". Durch das Auswählen einer Option aus dem zweiten Pulldown legt der Anwender eine Zeichenart fest. Für jede Zeichenart gibt es ein eigenes Unterprogramm, das angesprungen wird, sobald die linke Maustaste betätigt wird. Was dann weiter passiert, ist Sache des aufgerufenen Unterprogramms.

Hat 'Modus' den Wert 2, ist zur Zeit der Programmteil aktiv, in dem die Farben für Zeichnen und Ausmalen ausgewählt werden können. In diesem Fall wird die Kontrolle dem Unterprogramm 'Farbeinstellung:' übergeben. Um dem aufrufenden Programmteil mitzuteilen, ob die Farbeinstellung beendet ist (der Anwender erreicht das durch Klicken in ein OK-Feld), gibt es die Variable 'EndOK'. Ist 'EndOK'=1, führt AmigaBASIC den Programmteil 'Farbaus:' aus.

Beim Modus 3 passiert im Prinzip dasselbe wie bei Modus 2, nur geht es diesmal um den Teil, der für die Definition der Füllmuster zuständig ist. Das Unterprogramm 'Musterdef:' wird aufgerufen. Soll die Musterdefinition beendet werden, muß 'EndOK'=2 sein.

Modus 4 unterscheidet sich auch nicht weiter von seinen drei Artgenossen. Hierbei dreht es sich um eine besondere Funktion des Farb-Unterprogramms. Es wird keine Farbe ausgewählt, sondern mit Schiebereglern der R-, G- und B-Anteil einer Farbe festlegt. Das zuständige Unterprogramm heißt 'RGBDef:'. Damit die Farbdefinition (Festlegen der Farben mit RGB-Reglern) von der Farbauswahl (Aussuchen der Farben, die zum Zeichnen und Ausmalen benutzt werden) unterschieden werden kann, gibt es dafür einen eigenen Modus. Wenn dieser Modus durch 'EndOK'=3 beendet wurde, kehrt das Programm zunächst zum Modus 2 zurück, also zur Farbauswahl.

Weil 'Maus:' ein Unterprogramm ist, muß es mit einem RETURN-Befehl beendet werden, der zum aufrufenden Programmteil zurückspringt.

Der nächste Programmteil namens 'Projekt:' kümmert sich um die Funktionen des ersten Pulldowns. Die Variable 'Menpunkt' enthält den Menüpunkt, der von 'Menuauswahl:' festgestellt wurde. Abhängig vom Wert 'Menpunkt' werden jetzt die Programme aufgerufen, die die verschiedenen Funktionen ausführen.

Menüpunkt 1 wird dazu verwendet, aus dem Farb- oder dem Muster-Unterprogramm zum Zeichenmodus zurückzukehren. Am Anfang des Programms ist der Zeichenmodus automatisch aktiv (in den Vorbereitungen heißt es ja nicht umsonst 'Modus'=1), er muß nicht extra gewählt werden. Also braucht das Programm beim ersten Menüpunkt nur die Teile aufzurufen, die das Farb- und das Muster-Unterprogramm beenden, nämlich 'Farbaus:' und 'Musteraus:'

Menüpunkt 2 ruft das Farbauswahl-Unterprogramm auf. Für den Fall, daß das Musterdefinitions-Programm schon läuft, wird es zuerst durch GOSUB Musteraus abgeschaltet. Das zweite Pulldown, das die Zeichenarten anbietet, soll gesperrt werden, während das Farb-Programm läuft. Also: MENU 2,0,0. Die Variable 'Modus' erhält den Wert 2, dann wird das Label 'Farbauswahl:' angesprungen.

Der dritte Menüpunkt dient zum Aktivieren des Muster-Editors. In diesem Unterprogramm ist es möglich, Füllmuster zu definieren. Zuerst wird das eventuell noch aktive Farb-Programm abgeschaltet, wieder die Zeichenarten-Auswahl gesperrt und 'Modus' auf 3 gesetzt. Nach diesen Vorbereitungen geht's weiter mit 'Mustereditor:'.

Die Menüpunkte 4 und 5 sind zur Zeit noch nicht verfügbar. Hier werden Sie Bilder auf Diskette abspeichern und wieder einlesen können. Diese Features wollen wir aber erst im nächsten Teil des Buches hinzufügen, da Sie dort auch Näheres über die

Arbeit mit Peripheriegeräten und Datei-Verwaltung erfahren werden. Bis dahin sollten Sie sich auch größere, wertvolle Kunstwerke verkneifen.

Menüpunkt 6 ist nur im Modus 1 erlaubt. Damit wird das aktuelle Bild gelöscht. Der Programmteil 'Abfrage:' bringt vorher ein Window auf den Bildschirm, in dem Sie noch einmal gefragt werden, ob Sie das aktuelle Bild wirklich verlieren wollen. Wenn Sie dieser Abfrage Ihr OK gegeben haben, tritt das Löschen in Aktion. Mit den Befehlen Adef=0 und AREAFILL entledigt sich das Malprogramm eines eventuell in Entstehung befindlichen Areas. Das ist nämlich die einzige Zeichenart, wo das Ergebnis nicht sofort entsteht, da für das Vieleck in mehreren Arbeitsgängen die einzelnen Ecken definiert werden. Dieser Vorgang würde ja durch das Bildschirmlöschen unterbrochen und muß deshalb an dieser Stelle beendet werden. CLS löscht schließlich den Bildschirm.

Der siebte und letzte Menüpunkt ermöglicht das Beenden des Malprogramms. Dazu werden zuerst die Farb- und Musterdefinition abgeschaltet, dann folgt wie beim Löschen eine Sicherheitsabfrage. Gibt sie grünes Licht, springt das Programm zum Label 'Ende:', wo der Ausstieg aus dem Malprogramm durchgeführt wird.

Auch das 'Projekt:'-Unterprogramm endet mit RETURN.

'Zeichenart:' kümmert sich um das zweite Menü. Hier werden die Werkzeuge zum Zeichnen ausgewählt. Der alte Menüpunkt verliert sein Häkchen, dann wird die Variable 'Zart' auf den neuesten Stand gebracht und schließlich der neue Menüpunkt mit einem Häkchen versehen. Das war schon alles.

Jetzt folgen die Unterprogramme, die die einzelnen Zeichenarten ermöglichen. Es geht los mit der Version des Freihandzeichnens, die eine dünne Linie erzeugt:

```
DueFreihand:
  Test=MOUSE(0) : x=MOUSE(1) : y=MOUSE(2)
  WHILE MOUSE(0)<>0
```

```
    LINE (x,y)-(MOUSE(1),MOUSE(2)),Zfarbe
    x=MOUSE(1) : y=MOUSE(2)
  WEND
RETURN
```

Vielleicht wundern Sie sich, warum wir den Wert von MOUSE(0) einer Variablen 'Test' zuweisen, die wir dann überhaupt nicht mehr benötigen. Dafür gibt es trotzdem einen guten Grund: Wie Sie sich erinnern, kann mit MOUSE(0) abgefragt werden, ob die linke Maustaste gedrückt ist oder gedrückt wurde. MOUSE(1) liefert die x- und MOUSE(2) die y-Koordinate des Mauszeigers. Nimmt man es ganz genau, und darauf kommt's in diesem Fall an, liefert MOUSE(1) die x-Koordinate des Punkts, wo die Maus beim letzten Aufruf der Funktion MOUSE(0) stand, und MOUSE(2) die entsprechende y-Koordinate. Um einen möglichst aktuellen Punkt zu bekommen, rufen wir also ganz kurz vor der Koordinaten-Abfrage die Funktion MOUSE(0) auf.

Die WHILE...WEND-Schleife verbindet den letzten verwendeten Punkt solange mit dem aktuellen Standpunkt des Mauszeigers, wie die Maustaste gedrückt bleibt. 'Zfarbe' ist übrigens die Zeichenfarbe. Das Resultat: Es entsteht ein durchgezogener Strich, Sie können mit gedrückter Maustaste freihandzeichnen. Das Unterprogramm für das Freihandzeichnen mit einem dicken Strich unterscheidet sich nur gering:

```
DiFreihand:
  Test=MOUSE(0)
  WHILE MOUSE(0)<>0
    x=MOUSE(1) : y=MOUSE(2)
    LINE (x,y)-(x+5,y+5),Zfarbe,bf
  WEND
RETURN
```

Um einen dicken Strich zu erzeugen, werden an der aktuellen Position des Mauszeigers kleine Blocks gezeichnet. Solange Sie die Maus langsam bewegen, entsteht ein dicker Strich. Wird die

Bewegung allerdings zu schnell, haben die Kästchen einen kleinen Abstand voneinander, eine Art gepunktete Linie entsteht.

Apropos gepunktete Linie: Wenn Sie so einen ähnlichen Effekt absichtlich verwenden wollen, können Sie das mit der nächsten Zeichenart:

```
Punkte:
  Test=MOUSE(0)
  WHILE MOUSE(0)<>0
    PSET (MOUSE(1),MOUSE(2)),Zfarbe
  WEND
RETURN
```

An der Position des Mauszeigers wird ein Punkt gezeichnet. Bei sehr langsamer Bewegung der Maus entsteht eine durchgezogene, bei normaler und schneller Bewegung eine gepunktete Linie. Diese Darstellung ist übrigens dieselbe, die wir in unserem ersten kleinen Beispielprogramm zur Maussteuerung (Sie wissen schon: Die Super-Mini-Version des Malprogramms) benutzt haben.

Auch ein Spraydosen-Effekt ist nicht schwer zu programmieren:

```
Spraydose:
  Test=MOUSE(0)
  WHILE MOUSE(0)<>0
    x=MOUSE(1)+14*RND : y=MOUSE(2)+7*RND
    LINE (x,y)-(x,y),Zfarbe,bf
  WEND
RETURN
```

Bei der Spraydose sollen innerhalb eines bestimmten Bereichs einzelne, verstreute "Farbspritzer" entstehen. Deshalb lassen wir die Punkt-Koordinaten durch Zufallszahlen berechnen. Sie können horizontal bis zu 14 und vertikal bis zu 7 Punkte von der Mausposition abweichen. Dorthin setzen wir dann einen Punkt.

Aber halt! Warum verwenden wir denn für einen simplen Punkt den komlizierten LINE-Befehl mit Blockfill-Option? Es entsteht ja tatsächlich nur ein einziger Punkt, denn die linke obere und die rechte untere Begrenzung des Rechtecks sind identisch. Wir haben das natürlich mit Absicht gemacht. Diese Methode hat nämlich gegenüber PSET einen großen Vorteil: Die Blockfill-Option bewirkt, daß auf jeden Fall das angegebene Füllmuster verwendet wird, egal wie groß das gezeichnete Objekt ist. Haben Sie also später ein bestimmtes Füllmuster ausgewählt und halten die Spraydose längere Zeit auf dieselbe Stelle, bildet sich das Muster. Damit lassen sich interessante Effekte erzielen.

Da wir jetzt schon gerade einmal bei LINE sind, kommen gleich noch mehr Funktionen, die diesen Befehl benutzen:

```
Linien:
  Test=MOUSE(0)
  x1=MOUSE(3) : y1=MOUSE(4)
  PSET (x1,y1),Zfarbe
  WHILE MOUSE(0)<>0
  WEND
  LINE (x1,y1)-(MOUSE(5),MOUSE(6)),Zfarbe
RETURN
```

Mit diesem Teil haben Sie die Möglichkeit, Geraden zwischen zwei Punkten zu zeichnen. Die Bedienung ist ganz einfach: Bewegen Sie den Mauszeiger an den Anfangspunkt, drücken Sie die linke Maustaste und schieben Sie die Maus nun mit gedrückter Maustaste an den Endpunkt. Dann lassen Sie los. Schon ist die Linie gezeichnet.

Sie haben sicher die neuen, Ihnen noch unbekannten Funktions-Argumente von MOUSE entdeckt: MOUSE(3) und MOUSE (4), MOUSE(5) und MOUSE(6) werden da verwendet. Diese Werte funktionieren so: Wird die Maustaste gedrückt, die Maus bewegt und dann die Maustaste wieder losgelassen, liefert:

MOUSE(3) die x-Koordinate des Anfangspunkts der Mausbewegung,

MOUSE(4) die zugehörige y-Koordinate,

MOUSE(5) die x-Koordinate des Endpunkts der Mausbewegung,

MOUSE(6) die zugehörige y-Koordinate.

Um den Unterschied noch mal klar zu machen: Mit MOUSE(1) und MOUSE(2) erhalten Sie einen bestimmten Punkt, eine Art Momentaufnahme des Augenblicks, an dem MOUSE(0) abgefragt wurde. MOUSE(3) bis MOUSE(6) hingegen geben Ihnen den Anfangs- und Endpunkt einer Bewegung an. Sie beginnt mit dem Drücken der Maustaste und endet mit dem Loslassen der Taste.

Der Anfangspunkt wird vom Programm als 'x1' und 'y1' gemerkt. Damit der Anwender den Anfangspunkt sehen kann, wird der Punkt mit PSET auf den Bildschirm gezeichnet. Die leere WHILE...WEND-Schleife wartet, solange die Maustaste gedrückt ist. Ist sie nicht mehr gedrückt (haben Sie also losgelassen), wird die Linie vom gespeicherten Anfangspunkt zum aktuellen Endpunkt gezeichnet.

Die Funktionen von MOUSE(3), MOUSE(4) und MOUSE(5), MOUSE(6) sind gerade beim Zeichnen von Linien und überhaupt allen Figuren, für die mehr als ein Punkt angegeben werden muß, sehr nützlich. Auch zum Zeichnen von Rechtecken können wir sie benutzen:

```
Rechtecke:
  Test=MOUSE(0)
  x1=MOUSE(3) : y1=MOUSE(4)
  Zeiger(0,0)=x1 : Zeiger(0,1)=y1
  Zeiger(1,0)=x1 : Zeiger(2,1)=y1
  Anzahl=4
  WHILE MOUSE(0)<>0
    Zeiger(3,0)=MOUSE(5)
    Zeiger(3,1)=MOUSE(6)
    Zeiger(1,1)=Zeiger(3,1)
    Zeiger(2,0)=Zeiger(3,0)
    GOSUB Zeigerpunkte
```

```
    WEND
    LINE (x1,y1)-(Zeiger(3,0),Zeiger(3,1)),Zfarbe,b
  RETURN
```

Der größte Teil dieser Routine aktualisiert den Inhalt eines Datenfelds namens 'Zeiger'. Erst ganz am Schluß steht ein LINE-Befehl mit Block-Option. Was soll das schon wieder?

Bei manchen Funktionen, wie dem Zeichnen von Rechtecken, ausgemalten Rechtecken oder Kreisen, kann der Anwender vor der endgültigen Darstellung eines Objekts dessen Lage und Größe festlegen. Während dieses Vorgangs muß ihm angezeigt werden, wie die tatsächlichen Ausmaße gerade sind. Professionelle Zeichenprogramme zeigen das entstehende Objekt, also zum Beispiel das Rechteck, ständig in Bewegung. Das Rechteck wird größer und kleiner, erscheint dabei immer auf dem Bildschirm, läßt den Hintergrund aber unverändert. Diese Möglichkeit verwenden wir in unserem Malprogramm nicht, da Amiga-BASIC für solche Funktionen erstens zu wenig Speicherplatz bietet und zweitens in punkto Geschwindigkeit ganz schön zu kämpfen hätte.

Wir haben eine andere Lösung gefunden: Unser Programm zeigt einfach nur die Eckpunkte des Objekts an. Diese Eckpunkte befinden sich im Feld 'Zeiger', und zwar in dieser Reihenfolge: 'Zeiger(0,0)' enthält die x-Koordinate des ersten Punktes, 'Zeiger(0,1)' die zugehörige y-Koordinate. In 'Zeiger(1,0)' steht der x-Wert des zweiten Punktes, in 'Zeiger(1,1)' der zugehörige y-Wert, in (2,0) der dritte x-Wert, in (2,1) der dritte y-Wert usw. Die Variable 'Anzahl' speichert die Anzahl der Punkte. Das Unterprogramm 'Zeigerpunkte:' bringt dann die Werte aus 'Zeiger' als bewegliche Punkte auf den Bildschirm.

Um ein Rechteck zu zeichnen, drückt der Anwender die Maustaste und markiert damit einen Eckpunkt. Nun kann er mit gedrückter Maustaste den anderen, diagonal gegenüberliegenden Eckpunkt plazieren, die Zeigerpunkte geben dabei ständig die anderen Eckpunkte des aktuellen Rechtecks an. Nach dem Loslassen der Maustaste wird das Rechteck gezeichnet.

Noch etwas ist auffällig: Innerhalb der WHILE...WEND-Schleife fragen wir MOUSE(5) und MOUSE(6) ab, obwohl doch die Maustaste noch gedrückt ist, die Bewegung also noch gar nicht abgeschlossen ist. Wenn Sie MOUSE(5) und MOUSE(6) benutzen, bevor die Maustaste losgelassen wurde, erhalten Sie denselben Wert wie mit MOUSE(1) und MOUSE(2). Das heißt, dann ist wieder der Zeitpunkt des letzten Aufrufs der Funktion MOUSE(0) ausschlaggebend für die Koordinaten, die als Ergebnis geliefert werden.

Das Teilprogramm für das Zeichnen von Blocks ist bis auf ein einziges Zeichen völlig identisch mit dem Teil für Rechtecke: Der LINE-Befehl wird nicht mit der Option ,b, sondern mit der Option ,bf (Blockfill) verwendet. Am besten kopieren Sie mit "Copy" und "Paste" aus dem "Edit"-Pulldown den letzten Programmteil unter das bisherige Listing und machen die notwendigen Änderungen. Das gilt natürlich nur dann, wenn Sie das Programm selbst abtippen, statt die Version auf der beiliegenden Diskette zu nutzen.

```
Block:
  Test=MOUSE(0)
  x1=MOUSE(3) : y1=MOUSE(4)
  Zeiger(0,0)=x1 : Zeiger(0,1)=y1
  Zeiger(1,0)=x1 : Zeiger(2,1)=y1
  Anzahl=4
  WHILE MOUSE(0)<>0
    Zeiger(3,0)=MOUSE(5)
    Zeiger(3,1)=MOUSE(6)
    Zeiger(1,1)=Zeiger(3,1)
    Zeiger(2,0)=Zeiger(3,0)
    GOSUB Zeigerpunkte
  WEND
  LINE (x1,y1)-(Zeiger(3,0),Zeiger(3,1)),Zfarbe,bf
RETURN
```

Mit der Block-Option zeichnen Sie dieselben Rechtecke wie vorhin, aber diesmal ausgefüllt. Zum Ausfüllen wird das aktuelle Füllmuster verwendet. Um ausgefüllte Flächen zu bekommen,

kennen wir ja nicht nur Blocks, sondern auch den AREA-Befehl. Er darf natürlich in unserem Malprogramm nicht fehlen.

```
Flaechen:
  Test=MOUSE(0)
  x1=MOUSE(3) : y1=MOUSE(4)
  IF y1>186 THEN y1=186
  IF x1>311 THEN x1=311
  AREA (x1,y1)
  IF Adef=0 THEN Adef=1 : xa=x1 : ya=y1
  IF Adef<>1 AND x1=xa AND y1=ya THEN Fertig
  Adef=Adef+1 : IF Adef=20 THEN Fertig
  LINE (xa,ya)-(x1,y1),Zfarbe
  xa=x1 : ya=y1
RETURN

Fertig:
  Adef=0 : COLOR Zfarbe,0 : AREAFILL
RETURN
```

So funktioniert das Zeichnen von Areas: Nachdem Sie den Punkt "Vielecke" im Zeichenart-Pulldown gewählt haben, positionieren Sie den Mauszeiger auf den ersten Eckpunkt, den das Vieleck haben soll. Klicken Sie dann einmal mit der linken Maustaste und bewegen Sie die Maus zum zweiten Eckpunkt. Die bisher definierten Punkte werden durch Linien miteinander verbunden, so daß Sie bereits beim Entstehen der Figur die späteren Umrisse verfolgen können. Sie können das Ganze beliebig oft wiederholen. Das heißt, fast beliebig oft... Nach 20 Punkten wird das Vieleck automatisch auf den Bildschirm gebracht, da sich AmigaBASIC beim AREA-Befehl nicht mehr Punkte merken kann. Und was müssen Sie tun, wenn Sie das Vieleck vor dem zwanzigsten Punkt beenden wollen? Das Vieleck erscheint sofort auf dem Bildschirm, wenn Sie zweimal hintereinander die linke Maustaste drücken, ohne die Maus zwischen den beiden Klicks zu bewegen.

Der AREA-Befehl würde eine Fehlermeldung bringen, wenn ein Punkt außerhalb des erlaubten Bereichs angegeben würde. Der erlaubte Bereich ist in unserem Window 186 Punkte hoch und

311 Punkte breit. Die Werte werden vom 'Flaechen:'-Unterprogramm automatisch in diesen Bereich eingeschränkt. (Die beiden IF-Zeilen sind dafür zuständig.) Danach teilt das Programm die Punkte des aktuellen Klicks mit AREA dem Computer mit. Die Variable 'Adef' speichert die Anzahl der bereits definierten Ecken. Falls 'Adef' gleich 0 ist, wird es auf 1 gesetzt, und AmigaBASIC speichert die Koordinaten, die die Maus beim Aufruf hatte, in den Variablen 'xa' (für x alt) und 'ya' (für y alt). Die beiden brauchen wir, um vom letzten verwendeten Punkt eine Linie zum neuen Punkt zu ziehen. Unsere nächste Programmzeile prüft ab, ob sich die x-Koordinate oder die y-Koordinate seit dem letzten Klick verändert haben. Falls nicht, ist die Definition zu Ende ('Fertig:'). Bei jeder Angabe eines Punkts zählt AmigaBASIC die Variable 'Adef' hoch. Hat 'Adef' den Wert 20 erreicht, wird durch Aufruf des Unterprogramms 'Fertig:' ebenfalls mit der Definition Schluß gemacht. Zuletzt merken sich 'xa' und 'ya' noch die neuen alten Werte. Der 'Fertig:'-Teil setzt 'Adef' wieder auf 0, stellt mit einem COLOR-Befehl die verwendete Zeichenfarbe für den AREAFILL-Befehl ein und zeichnet das gewünschte Objekt. Fertig.

Auf zur nächsten Runde! Wir haben uns bisher mit allen möglichen Arten von eckigen Gebilden beschäftigt - es wird Zeit, mal eine runde Sache anzugehen. Dieser Teil zeichnet Kreise und Ellipsen:

```
Kreis:
  Test=MOUSE(0)
  x1=MOUSE(3) : y1=MOUSE(4)
  Zeiger(0,0)=x1 : Zeiger(0,1)=y1
  Zeiger(1,0)=x1 : Zeiger(2,1)=y1
  Zeiger(3,0)=x1 : Zeiger(4,1)=y1
  Anzahl=5
  WHILE MOUSE(0)<>0
    r1=ABS(x1-MOUSE(5))
    r2=ABS(y1-MOUSE(6))
    Zeiger(1,1)=y1-r2 : Zeiger(2,0)=x1+r1
    Zeiger(3,1)=y1+r2 : Zeiger(4,0)=x1-r1
    GOSUB Zeigerpunkte
  WEND
```

```
    IF r1=0 THEN r1=.1
    IF r1<r2 THEN Faktor=(r2/r1) : r1=r1*Faktor : r2=r2*Faktor
    CIRCLE (x1,y1),r1,Zfarbe,,,(r2/r1)
RETURN
```

Im Malprogramm geht das Zeichnen von Kreisen bzw. Ellipsen so: Bewegen Sie den Mauscursor auf den Mittelpunkt des gewünschten Kreises. Drücken Sie die rechte Maustaste und halten Sie sie gedrückt. Verschieben Sie nun die Maus. Der horizontale Abstand der Maus zum Mittelpunkt legt den x-Radius der Ellipse fest, der vertikale Abstand den y-Radius. An den Stellen, die am Kreis (an der Ellipse) am weitesten vom Mittelpunkt entfernt sind, sehen Sie vier Zeigerpunkte. Der fünfte Zeigerpunkt zeigt den Mittelpunkt an. Die Lage der einzelnen Punkte am Kreis ist in Bild 9 dargestellt.

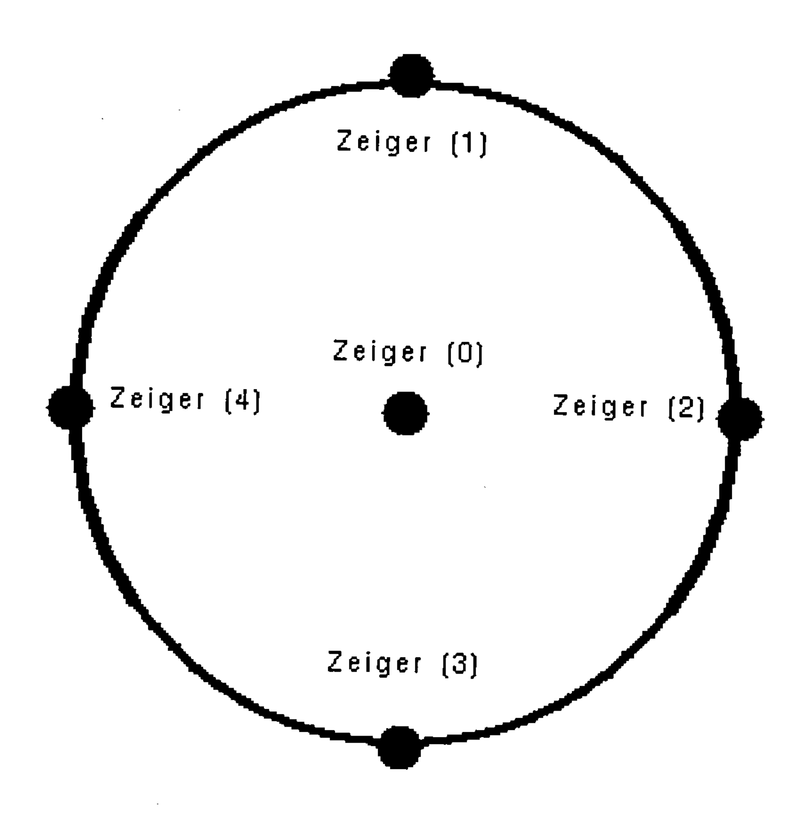


Bild 9: Die Lage der Zeigerpunkte an einem Kreis

Durch Loslassen der Maustaste wird der Kreis gezeichnet.

In 'x1'und 'y1' wird der Mittelpunkt festgelegt. Wenn Sie die obige Abbildung mit den Werten vergleichen, die dem Feld 'Zeiger' zugewiesen werden, werden Sie sicher schnell verstehen, wie's funktioniert: Jeder Punkt hat entweder den x-Wert oder den y-Wert seiner Koordinate mit dem Mittelpunkt gemeinsam. Der jeweils andere Wert wird in der WHILE...WEND-Schleife ermittelt. 'r1' ist der Absolutwert (also Vorzeichen-freie Wert) des x-Radius, 'r2' dasselbe für den y-Radius. Solange die Maustaste gedrückt bleibt, werden nur die Zeigerpunkte

aktualisiert und dargestellt. Falls der x-Radius gleich 0 ist, soll er auf 0.1 erhöht werden, da in den folgenden Berechnungen sonst ein "Division by Zero"-Error entstehen könnte.

Uns geht es jetzt noch um das Verhältnis der beiden Radius-Werte zueinander. Das brauchen wir ja, um bei CIRCLE einen Wert für das x/y-Verhältnis angeben zu können. Normalerweise funktioniert das Ganze nur, wenn der x-Radius größer oder gleich dem y-Radius ist. Trifft diese Bedingung nicht zu (IF r1<r2 ...), müssen wir das Verhältnis der beiden Werte umkehren. Ist auch das geschehen, folgt der CIRCLE-Befehl, der alle berechneten Daten schließlich in den gewünschten Kreis verwandelt.

Kommen wir nun zum Ausmalen von umrandeten Flächen. Das Stichwort heißt PAINT.

```
Fuellen:
Test=MOUSE(0)
IF Klick=0 THEN
  Klick=1
  SOUND 440,6,20
  x=MOUSE(1) : y=MOUSE(2)
  RETURN
ELSE
  Klick=0
  IF MOUSE(1)=x AND MOUSE(2)=y THEN
    PAINT (x,y),Ffarbe,Zfarbe
  ELSE
    SOUND 440,6,20
  END IF
END IF
RETURN
```

Hier tummelt sich zur Abwechslung mal wieder ein bißchen was Neues. Als erstes fällt Ihnen wahrscheinlich die ungewöhnliche Struktur auf, die die drei Befehle IF...THEN, ELSE und END IF zusammen bilden. Die Einzelbestandteile kennen Sie

eigentlich schon: IF...THEN ermöglicht das Auswerten von Bedingungen, und ELSE hinter IF...THEN gibt an, was getan werden soll, wenn die jeweilige Bedingung nicht zutrifft.

Oft sollen abhängig von einer Bedingung mehrere Befehle ausgeführt werden. Bisher mußten Sie dazu alles in eine Zeile schreiben. Solange es sich nicht um zuviele Befehle handelt, geht das noch, weil Programmzeilen im LIST-Window bis zu 255 Zeichen lang sein dürfen. Aber sehr übersichtlich ist es sicher nicht.

Deshalb haben Sie die Möglichkeit, den IF...THEN...ELSE-Befehl zu entzerren und zwischendrin beliebig lange Programmteile einzufügen. Wichtig ist nur, daß in derselben Zeile hinter IF...THEN und hinter ELSE kein Befehl mehr kommt. Denn daran unterscheidet AmigaBASIC die geschachtelte IF...THEN-Struktur von der normalen. Die auszuführenden Befehle beginnen einfach eine Zeile tiefer. Der ganze Block wird durch den Befehl END IF abgeschlossen. Die Struktur sieht also so aus:

```
IF (Bedingung) THEN
  (Programmteil, der bei
  "Bedingung trifft zu"
  ausgeführt wird.)
ELSE
  (Programmteil, der bei
  "Bedingung trifft nicht zu"
  ausgeführt wird.)
END IF
```

Wie Sie im Programm sehen, ist es auch erlaubt, IF...THEN/ELSE/END IF-Strukturen ineinander zu verschachteln. Wer dabei die Übersicht behalten will, sollte aber beim Schreiben des Programms unbedingt an die Einrückungen denken.

Warum bauen wir um den eigentlich einfachen PAINT-Befehl eine so komplizierte Umgebung auf? Erinnern Sie sich noch an den Object Editor? Er ist ja auch eine Art Malprogramm, geschrieben in AmigaBASIC. Wenn Sie beim Object Editor eine

umrandete Fläche ausmalen wollen, kann es Ihnen schnell passieren, daß Sie Ihr ganzes Bild zerstören. Entweder Sie haben die falsche Begrenzungsfarbe gewählt, oder Sie haben überhaupt vergessen, daß die PAINT-Funktion aktiv ist.

Nach dieser (auch für uns oft leidvollen) Erfahrung wollten wir unser Möglichstes dagegen tun, daß Ihnen beim Ausmalen in unserem Programm versehentlich etwas ähnliches passiert. Leider können wir das AmigaBASIC nicht verbessern, aber wir hatten eine Idee: Auf der Workbench dient ein Doppelklick dazu, eine Auswahl endgültig zu machen. Das wollten wir übernehmen.

Wenn Sie ausmalen wollen, müssen Sie also den Mauszeiger in die begrenzte Fläche bewegen und zweimal klicken. Zwischen den beiden Klicks darf die Maus nicht verschoben werden. Beim ersten Klick hören Sie einen Warnton (zum SOUND-Befehl kommen wir an anderer Stelle in diesem Buch). Falls Sie vergessen haben sollten, daß die Zeichenart "Ausmalen" aktiv ist, erinnert Sie der Ton daran. Sie können einfach eine andere Zeichenart wählen, und nichts ist passiert. Nur wenn Sie an derselben Position ein zweites Mal klicken, wird das Ausmalen wirklich durchgeführt.

Um festzustellen, beim wievielten Klick wir gerade sind, benutzen wir die Variable 'Klick', deren Name eigentlich schon alles sagt. Und so funktioniert der Programmteil: Beim ersten Aufruf ist 'Klick'=0. Der Anwender hat an der aktuellen Mausposition einmal geklickt. Nun wird 'Klick' auf 1 gesetzt, der Warnton erklingt, das Programm merkt sich die Koordinaten des Mauszeigers und springt zurück. Die Füllfarbe 'Ffarbe' und die Zeichen- bzw. Begrenzungfarbe 'Zfarbe' können Sie im Farbauswahl-Unterprogramm festlegen. Erst wenn der Anwender ein zweites Mal geklickt hat, wird die 'Fuellen:'-Routine wieder aufgerufen. Diesmal hat 'Klick' den Wert 1. Das Programm führt also den ELSE-Teil aus: 'Klick' wird wieder 0. Falls sich die Position der Maus nicht geändert hat (dazu vergleichen wir einfach die aktuellen Werte mit den in 'x' und 'y' gespeicherten), wird ausgemalt. Sind die Koordinaten jedoch nicht identisch, erklingt ein zweites Mal der Warnton und das Unterprogramm springt zurück. Soll dann doch ausgemalt werden, muß der An-

wender an der neuen Stelle zweimal klicken. Es gehört ein wenig Übung dazu, die Maus während des Klickes ganz ruhig zu halten, aber dieser kleine Mangel an Komfort ist vielleicht besser, als versehentlich das ganze Bild zu ruinieren.

Falls Sie trotzdem eine kleine Bewegung der Maus zulassen wollen, ändern Sie die zweite IF...THEN-Zeile im 'Fuellen:'-Unterprogramm wie folgt:

```
IF ABS(x-Mouse(1))<11 AND ABS(y-MOUSE(2))<6 THEN
```

Die Änderung erlaubt eine Abweichung von der alten Mausposition bis zu 10 Punkte horizontal und bis zu 5 Punkte vertikal. Natürlich steigt damit das Risiko, doch einmal einen Fehler zu machen.

Da wir jetzt mit allen Möglichkeiten vertraut sind, etwas aufs Papier bzw. auf den Bildschirm zu bringen, wollen wir noch etwas dafür tun, daß wir das Gemalte auch wieder entfernen können. Die Kunst besteht ja schließlich oft auch im Weglassen. Dazu gibt es den Radiergummi:

```
Radiergummi:
  Test=MOUSE(0)
  WHILE MOUSE(0)<>0
    x=MOUSE(1) : y=MOUSE(2)
    PATTERN ,Voll%
    LINE (x,y)-(x+10,y+5),0,bf
    PATTERN ,Muster%
  WEND
RETURN
```

Den Radiergummi benötigen Sie, um gezeichnete Stellen wieder vom Bildschirm zu entfernen. Das Programm funktioniert ganz genau so wie der Teil zum dicken Freihandzeichnen. Nur wird diesmal nicht die Zeichenfarbe, sondern die Farbe Nummer 0, also die Hintergrundfarbe, verwendet. Resultat: Die Stelle, an der mit dem Radiergummi gearbeitet wird, wird gelöscht.

Aber ein unbekannter Befehl tritt doch noch auf: PATTERN. "Pattern" heißt auf Deutsch "Muster". Es geht also wieder einmal um die schon öfter erwähnten Füllmuster. Die Erklärungen, wie der PATTERN-Befehl genau funktioniert, wollen wir uns noch etwas aufheben. Aber so viel sollten Sie jetzt schon wissen: PATTERN ,Voll% bewirkt in unserem Programm, daß ein volles, gefülltes Muster verwendet wird. PATTERN ,Muster% dagegen aktiviert das im Programmteil 'Mustereditor:' ausgewählte Muster.

Vor dem Befehl, der den Radiergummi auf den Bildschirm bringt, wählen wir das volle Muster, weil sonst nur mit dem aktuellen Muster gelöscht würde. Ergebnis wäre eine Negativdarstellung dieses Musters, mehr nicht. Wir wollen aber, daß wirklich alles an der betreffenden Stelle verschwindet. Das Gegenteil von "alles da" ('Voll%') ist "alles weg", deshalb die beiden PATTERN-Befehle. Es ist schon erstaunlich, woran man beim Programmieren alles denken muß. Aber keine Sorge: Auch wir haben nicht alles von Anfang an so gemacht. Hinter all diesen weisen Ratschlägen und Begründungen stecken -zig Testläufe und mindestens dreimal soviele "Warum funktioniert der Kram nicht?!"-Ausrufe.

Das nächste Unterprogramm, das mit den Zeichenarten zu tun hat, heißt 'Zeigerpunkte:'. Dieser Teil bringt die blinkenden Zeigerpunkte auf den Bildschirm, die von 'Rechtecke:', 'Block:' und 'Kreis:' im Feld 'Zeiger' übergeben werden.

```
Zeigerpunkte:
  FOR x=0 TO Anzahl-1
    xz=Zeiger(x,0) : yz=Zeiger(x,1)
    IF xz<0 THEN xz=0 :Zeiger(x,0)=0
    IF xz>311 THEN xz=311 : Zeiger(x,0)=311
    IF yz<0 THEN yz=0 : Zeiger(x,1)=0
    IF yz>186 THEN yz=186 : Zeiger(x,1)=186
    Altfarbe(x)=POINT(xz,yz)
  NEXT x
  FOR x=0 TO Anzahl-1
```

```
    PSET (Zeiger(x,0),Zeiger(x,1)),-(Altfarbe(x)=0)
  NEXT x
  FOR x=0 TO Anzahl-1
    PSET (Zeiger(x,0),Zeiger(x,1)),Altfarbe(x)
  NEXT x
RETURN
```

Der ganze Teil besteht aus drei FOR...NEXT-Schleifen. Die erste errechnet die Punkte und merkt sich, was dort vorher war. Die zweite Schleife setzt die Punkte, die dritte löscht sie wieder. Die Schleifen werden sooft durchlaufen, wie Punkte vorhanden sind. Da die Zählung bei 0 beginnt, hat der letzte Punkt die Nummer 'Anzahl'-1. Die vier IF...THEN-Zeilen in der ersten Schleife prüfen ab, ob die Koordinaten des aktuellen Punkts außerhalb des erlaubten Bereichs liegen. Falls der x-Wert kleiner 0 oder größer 311 oder falls der y-Wert kleiner 0 oder größer 186 ist, werden die Werte auf den jeweiligen noch erlaubten Grenzwert gesetzt.

Als nächstes merkt sich das Programm im Datenfeld 'Altfarbe' die Farbe des Punktes, der sich vorher an der Stelle befand, wo der Zeigerpunkt plaziert werden soll. Wenn Sie herausfinden wollen, welche Farbe ein bestimmtes Pixel hat, gibt es dazu die BASIC-Funktion POINT(x,y). Sie liefert die Farbnummer des angegeben Punkts, bzw. die Zahl -1, wenn der Punkt außerhalb des erlaubten Bereichs liegt.

Die einzige Aufgabe der zweiten Schleife ist, den aktuellen Punkt auf den Bildschirm zu bringen. Nur die verwendete Zeichenfarbe ist vielleicht noch erklärenswert: Sie wird berechnet durch den Ausdruck -(Altfarbe(x)=0). Der Grund ist ganz einfach: Wir wollen ja, daß sich die Zeigerpunkte auf jeden Fall deutlich vom Hintergrund abheben. Hatte der Punkt, der sich vorher an der Stelle befand, wo unser Zeigerpunkt erscheinen soll, die Hintergrundfarbe (Nummer 0, in der Grundeinstellung schwarz), dann soll der Zeigerpunkt in der Farbe Nummer 1 (weiß) gezeichnet werden. Hatte der alte Punkt irgendeine andere Farbe, soll der Zeigerpunkt die Hintergrundfarbe annehmen.

Jetzt wissen Sie zwar, was die Formel erreichen soll, aber immer noch nicht, wie sie funktioniert. Das ganze hängt mit der Methode zusammen, wie AmigaBASIC logische Vergleiche durchführt. Eine falsche Aussage hat den logischen Wert 0, eine wahre, zutreffende Aussage hat den Wert -1. Wir werden darauf im Zwischenspiel 4 noch genauer eingehen. Sie können's aber gleich mal im BASIC-Window ausprobieren:

```
? (0=1)
```

Frei nach Schneewittchen übersetzt, heißt das: "Lieber Amiga, sag' mir, ist Null gleich Eins?" Ihr Amiga wird Ihnen antworten: "Nein, lieber Besitzer, Null ist nicht gleich Eins." In seiner Sprache heißt das:

```
0
```

Wir finden allerdings die Grimm'sche Frage- und Antwortversion - mit Verlaub - sympathischer. Lautet die Frage jedoch

```
? (1=1)
```

dann heißt die Antwort

```
-1
```

also "Ja, lieber Besitzer, Eins ist gleich Eins." Wahre Aussagen haben den logischen Wert -1.

Die Formel -(Altfarbe(x)=0) liefert als Ergebnis:

0 wenn 'Altfarbe(x)' nicht gleich Null ist. Denn -(falsche Aussage) ist -0, also 0.

1 wenn 'Altfarbe(x)' gleich Null ist. Denn -(wahre Aussage) ist -(-1), also 1.

Sie sehen, manchmal steckt hinter einer unscheinbaren Formel eine ziemlich komplizierte Idee. Albert Einstein meinte zum

Beispiel nur: $e = mc^2$. Was er damit ausdrücken wollte, ist ein bißchen schwieriger zu erklären.

P.S.: Lieber Albert. Natürlich ist auch dieser Vergleich, wie alles, relativ.

Die letzte FOR...NEXT-Schleife versorgt die Pixels mit ihren alten Farben, die Zeigerpunkte verschwinden wieder vom Bildschirm.

Kommen wir jetzt zum letzten Unterprogramm, das noch zu den Zeichenarten gehört. Von "Zeichnen" kann man in diesem Fall eigentlich gar nicht mehr sprechen, es geht nämlich um Texteingaben:

```
Text:
  Test=MOUSE(0)
  x=MOUSE(1) : y=MOUSE(2)
  MENU OFF : MOUSE OFF
  MENU 1,0,0 : MENU 2,0,0
  WINDOW 5,"Bitte Text eingeben:",(0,177)-(311,185),18,1
  CLS
  LINE INPUT Text$
  WINDOW CLOSE 5
  WINDOW 2
  MENU 1,0,1 : MENU 2,0,1
  MENU ON : MOUSE ON
  LOCATE INT(y/8.17)+1,INT(x/7.94)+1 : COLOR Zfarbe,Ffarbe
  PRINT Text$;
  COLOR Zfarbe,0
RETURN
```

Für Beschriftungen in Ihren Bildern können Sie diesen Menüpunkt verwenden. Klicken Sie mit dem Mauscursor an die Stelle, wo der Text anfangen soll. Dann erscheint am unteren Bildschirmrand ein Window. Sie können dieses Window beliebig verschieben, wenn es Sie an seiner alten Position stören sollte. Geben Sie den gewünschten Text ein, gefolgt von <RETURN>. Daraufhin verschwindet das Window wieder, und der Text erscheint an der gewählten Stelle in Ihrem Bild.

Das Unterprogramm 'Text:' stellt zuerst die Mauskoordinaten fest. Während das Window zur Texteingabe auf dem Bildschirm steht, werden Event Trapping und die beiden Pulldowns gesperrt. Den Text lesen wir mit LINE INPUT ein, das heißt, Sie können beliebige Zeichen verwenden. Nach der Eingabe lassen wir das Window Nummer 5 wieder verschwinden und aktivieren Window 2. Die beiden Pulldowns und Event Trapping werden wieder zugelassen. Bevor der Text ins Bild geschrieben werden kann, muß das Programm die Pixel-Koordinaten in Zeilen und Spalten für LOCATE umrechnen. Bei den Farben für die Textdarstellung gibt es noch eine kleine Besonderheit: Um die Verwendung von Farben bei der Text-Option etwas flexibler zu gestalten, verwenden wir die aktuelle Zeichenfarbe 'Zfarbe' für den Text und die Ausmalfarbe 'Ffarbe' für den Untergrund. So können Sie farbige Texte schreiben, die von farbigen Balken hinterlegt sind. Wenn Sie keine Hinterlegung wünschen, wählen Sie einfach die Hintergrundfarbe (Nummer 0) als Ausmalfarbe.

Bevor wir das Unterprogramm verlassen, müssen wir mit COLOR Zfarbe,0 die Hintergrundfarbe wieder auf den Wert 0 setzen. Sonst würde z.B. beim Löschen des Bildschirms der ganze Schirm in der Ausmalfarbe gefüllt werden. Zum Thema Zeichnen haben wir alles erledigt. Vergessen Sie bloß nicht das Abspeichern zwischendurch...

Weiter geht's mit dem Farb-Unterprogramm. Es wird aufgerufen, wenn Sie den Punkt "Farben ändern" aus dem "Programm"-Pulldown anwählen. Das Farb-Programm gliedert sich in zwei Teile. Zum einen den Teil, wo Sie die Farben zum Zeichnen auswählen, und zum anderen den Teil, wo Sie die RGB-Zusammensetzung der einzelnen Farben verändern. Wir wollen uns zunächst den ersten Teil ansehen:

```
Farbauswahl:
  ZFWahl=0 : EndOK=0
  MOUSE OFF : MENU OFF
  WINDOW 3,"Farbmenü",(4,20)-(245,160),18,1
  PATTERN ,Voll%
  FOR x=1 TO (Maxfarbe+1)/8
    FOR y=0 TO 7
```

```
      LINE (y*30,(x-1)*16)-((y+1)*30,x*16),(x-1)*8+y,bf
    NEXT y
  NEXT x
  LINE (10,72)-(50,95),Zfarbe,b
  LINE (15,75)-(45,93),Zfarbe,bf
  LOCATE 13,2 : COLOR 0,1 : PRINT "Zeichn";
  LINE (70,72)-(110,95),Ffarbe,b
  LINE (75,75)-(105,93),Ffarbe,bf
  LOCATE 13,9 : COLOR 1,0 : PRINT "Ausmal";
  LINE (135,72)-(235,95),1,b
  LOCATE 11,21 : PRINT "Palette";
  LINE (190,109)-(230,132),1,b
  LOCATE 16,26 : PRINT "OK";
  PATTERN ,Muster%
  MOUSE ON : MENU ON
RETURN

Farbeinstellung:
  Test=MOUSE(0)
  x=MOUSE(3) : y=MOUSE(4)

  GOSUB Farbabfrage

  PATTERN ,Voll%
  LINE (10,72)-(50,95),Zfarbe,b
  LINE (15,75)-(45,93),Zfarbe,bf
  LINE (70,72)-(110,95),Ffarbe,b
  LINE (75,75)-(105,93),Ffarbe,bf
  PATTERN ,Muster%

  IF WINDOW(0)=3 AND 72<y AND y<95 THEN
    IF 70<x AND x<110 THEN ZFWahl=1
    IF 10<x AND x<50 THEN ZFWahl=0
    IF 135<x AND x<235 THEN
      PATTERN ,Voll%
      PAINT (137,74),3,1
      PATTERN ,Muster%
      GOSUB PaletteDef
      RETURN
```

```
      END IF
    END IF
    GOSUB OKCheck

    IF ZFWahl=0 THEN
      LOCATE 13,2 : COLOR 0,1 : PRINT "Zeichn";
      LOCATE 13,9 : COLOR 1,0 : PRINT "Ausmal";
    ELSE
      LOCATE 13,2 : COLOR 1,0 : PRINT "Zeichn";
      LOCATE 13,9 : COLOR 0,1 : PRINT "Ausmal";
    END IF
  RETURN

  Farbabfrage:
    IF WINDOW(0)=3 AND x<240 AND y<(2^(Farben+1)) THEN
      fx=INT(x/30) : fy=INT(y/16)
      IF ZFWahl=0 THEN
        ZFarbe=fy*8+fx
      ELSE
        Ffarbe=fy*8+fx
      END IF
    END IF
  RETURN

  OKCheck:
    IF x>190 AND x<230 AND y>109 AND y<132 THEN
      PATTERN ,Voll%
      PAINT (192,111),3,1 : EndOK=1
      PATTERN ,Muster%
    END IF
  RETURN

  Farbaus:
    MENU 2,0,1 : Modus=1
    WINDOW CLOSE 3
    WINDOW OUTPUT 2
  RETURN
```

Ein ganz schöner Otto, was? Wie üblich gehen wir ihn Stück für Stück durch. Wenn Sie das Farb-Programm aufrufen, erscheint zunächst ein neues Window auf dem Bildschirm, das deutlich kleiner ist als das Window, in dem Sie Ihr Bild zeichnen. Im neuen Window sehen Sie farbige Felder, eines pro verfügbarer Farbe. Ein Klick in das gewünschte Farbfeld, schon haben Sie eine neue Zeichenfarbe ausgewählt.

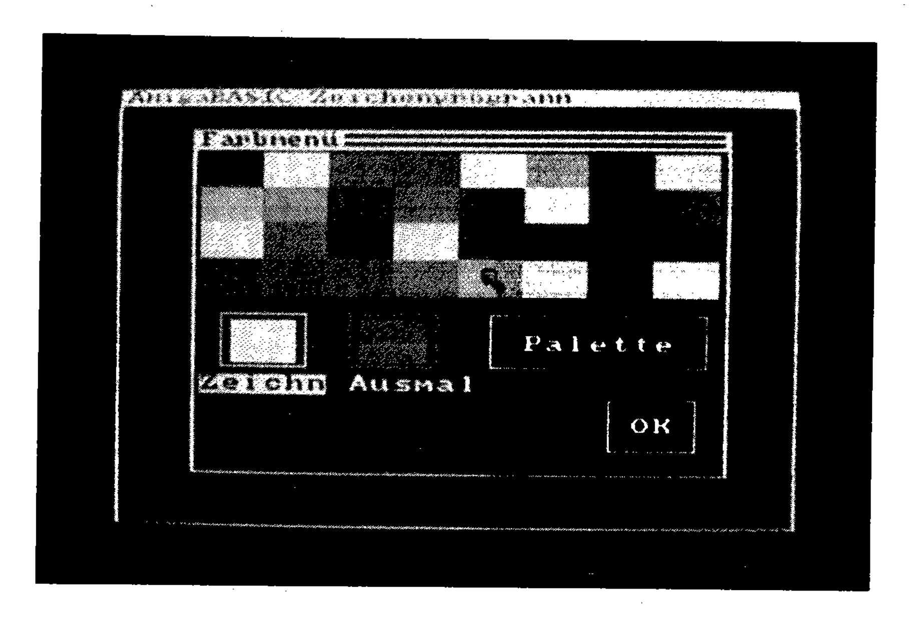


Bild 10: Das "Farbmenü"-Window aus dem Malprogramm

Unterhalb der Farbfelder gibt es noch vier weitere Kästchen. In der ersten Reihe sind zwei Felder, die die aktuelle Zeichenfarbe und die aktuelle Ausmalfarbe anzeigen. Die Ausmalfarbe ist nur für den PAINT-Befehl wichtig, da man für ihn eine Begrenzungsfarbe und eine Füllfarbe angeben kann.

Wenn Sie in eines dieser beiden Felder klicken, entscheiden Sie, ob Sie die Zeichenfarbe oder die Ausmalfarbe verändern wollen.

Die Schrift unter dem ausgewählten Kästchen ist hell hinterlegt. Die Kästchen selbst werden in der aktuellen Farbe dargestellt.

Das Feld "Palette" aktiviert später die Farb-Definition, zur Zeit ist dieser Programmteil noch nicht vorhanden. Ein Klick ins "OK"-Feld beendet das Farb-Unterprogramm.

Und wie haben wir das alles programmiert?

Das Unterprogramm 'Farbauswahl:' baut das Window und seinen Inhalt auf und trifft die nötigen Vorbereitungen. Die Variablen 'ZFWahl' und 'EndOK' werden auf 0 gesetzt. 'ZFWahl' gibt an, ob die Zeichenfarbe ('ZFWahl'=0) oder die Hintergrundfarbe ('ZFWahl'=1) geändert werden soll. 'EndOK' teilt dem aufrufenden Programmteil mit, ob die Farbauswahl beendet ist.

Während ein Window aufgebaut wird, sollte Event Trapping ausgeschaltet werden. Wechselt nämlich mitten im Aufbau das Ausgabe-Window, würde ein Teil der Texte und Grafiken im falschen Window landen. Wir unterbrechen deshalb die Überwachung von MOUSE und MENU mit den Befehlen MOUSE OFF und MENU OFF. Nun erscheint das neue Window auf dem Bildschirm. Es kann verschoben werden, und es merkt sich seinen Inhalt.

Für das Zeichnen der Farbkästchen brauchen wir wieder das volle einfarbige Muster. Deshalb der Befehl PATTERN ,Voll%. Die nächsten beiden, ineinander verschachtelten FOR...NEXT-Schleifen bringen die farbigen Felder ins Window. Ist die Anzahl der möglichen Farben 8 (3 Bit-Ebenen), wird eine Reihe mit 8 Feldern gezeichnet. Bei 16 möglichen Farben (4 Bit-Ebenen) entstehen zwei Reihen und bei 32 Farben vier Reihen. Die Formel '(Maxfarbe+1)/8' liefert bei 3 Bit-Ebenen als Ergebnis die Zahl 1, bei 4 Bit-Ebenen die Zahl 2 und bei 5 Bit-Ebenen die Zahl 4.

Im Anschluß daran bauen wir noch die beiden Kästchen für Zeichenfarbe und Ausmalfarbe auf und drucken den zugehörigen Text darunter. Auch die Kästchen für "Palette" und "OK" entstehen in diesem Unterprogramm. Vor dem Rücksprung wird

das Füllmuster zurück auf das gerade gewählte Muster geschaltet und Event Trapping wieder aktiviert.

Solange der Modus 2 aktiv ist, ist das Unterprogramm 'Farbeinstellung:' für die Auswertung von Mausklicks zuständig. Zu Beginn stellt dieser Programmteil die Koordinaten der Maus beim Aufruf fest. Dann wird erst mal das Unterprogramm 'Farbabfrage:' aufgerufen. Es wertet aus, ob in eines der Farbfelder geklickt wurde. Wenn ja, verwendet es die gewählte Farbe, je nach dem Wert von 'ZFWahl', entweder als neue Zeichen- oder als neue Ausmalfarbe. Wenn das Programm aus 'Farbabfrage:' zurückgekehrt ist, zeichnet 'Farbeinstellung:' die Kästchen für Zeichen- und Ausmalfarbe neu, damit ein eventueller Farbwechsel sichtbar wird. Dazu verwendet es wieder das Füllmuster aus dem Datenfeld 'Voll%'. Immer, wenn Kästchen oder Flächen außerhalb des Bildes gezeichnet werden, schalten wir mit PATTERN zurück auf dieses Voll-Muster, da die Flächen sonst in einem der Füllmuster gezeichnet würden.

Vielleicht verstehen Sie bisher nur "Füll" und "Voll" und fragen sich, wo eigentlich der Unterschied liegt: Es dauert nicht mehr lange, bis wir das Füllmuster-Unterprogramm besprechen. Dann lösen sich all diese Probleme in kleine nette Muster auf, die man ohne Schwierigkeiten versteht.

Die Aufgabe des Unterprogramms 'Farbabfrage:' haben wir schon erklärt: Es überprüft, ob in eines der Farbfelder geklickt wurde. Wenn ja, ändert es die Zeichen- oder die Ausmalfarbe. In der IF-Bedingung kommt die Funktion WINDOW(0) vor. Ähnlich wie bei MOUSE können auch verschiedene Daten über das aktuelle WINDOW abgefragt werden. WINDOW(0) liefert die Nummer des gewählten Windows. Das ist das Window, das der Anwender durch Klicken aktiviert hat. Es muß mit überprüft werden, da der Anwender ja auch in ein anderes Window als unser Farb-Window geklickt haben könnte. Sie kennen ja das nächteraubende Programmiererprinzip: Alle nur erdenklichen Fehler vor dem Anwender zu machen...

Das Unterprogramm soll nur ausgeführt werden, wenn der Punkt, der angeklickt wurde, sich innerhalb des Windows Nummer 3 und innerhalb des Bereichs befindet, wo die Farbfelder liegen. Die y-Koordinate vergleichen wir mit der Formel (2^(Farben+1)), die von der Anzahl der zulässigen Farben abhängt. Denn die untere Grenze bei nur einer Farb-Reihe unterscheidet sich von der Grenze, wenn zwei Reihen oder vier Reihen gezeigt werden. Gibt es nur eine Reihe, muß y kleiner 16 sein. Sind es zwei Reihen, ist 32 die Grenze. Bei vier Reihen, ist für y bei 64 Schluß. Die Farbkästchen sind genau 16 Pixels hoch. In 'Farben' steht die Anzahl der Bitebenen. Wenn Sie der Formel nicht trauen, setzen Sie einfach die möglichen Werte ein (3,4 oder 5) und rechnen Sie das Ergebnis aus. Sie sehen, es klappt. In der Variablen 'fx' berechnen wir, das wievielte Kästchen von links angeklickt wurde. Und in 'fy' steht, in welcher Reihe es liegt. Aus den beiden Werten kann die Farbnummer errechnet werden (Formel: 'fy*8+fx').

Je nach dem Wert von 'ZFWahl' wird die angeklickte Farbe der Zeichenfarbe 'Zfarbe' oder der Füllfarbe 'Ffarbe' zugewiesen. Der Label-Name 'OKCheck:' läßt schon vermuten, was der nächste Teil macht: Er prüft, ob ins OK-Feld geklickt wurde. Wenn ja, malen wir das OK-Feld orange aus, setzen 'EndOK' auf 1 und springen mit dem RETURN-Befehl zurück. Das Farbprogramm wird dann beendet. Und zwar durch das letzte Unterprogramm der Farbauswahl: 'Farbaus:'. Hier wird die Farbauswahl abgeschaltet. Das "Zeichenart"-Pulldown darf wieder benutzt werden, 'Modus' ist Nummer 1 (Zeichnen). Das Farb-Window (Nummer 3) verschwindet vom Bildschirm, und Window 2, in dem die Zeichnung erscheint, ist wieder Ausgabe-Window.

Wenn Sie das alles eingegeben haben, können Sie beim Zeichnen und Ausmalen schon aus 8, 16 oder 32 festeingestellten Farben wählen. Vielleicht gefallen Ihnen die Farben jedoch nicht, oder Sie brauchen für Ihr Landschaftsbild viele Grün-Töne und können auf orange, dunkelblau und rosa dankend verzichten. Oder Sie machen Werbung für die Bundesbahn und brauchen für Ihre rosa Elefanten nur einige Rosatöne. Wie auch immer, mit dem nächsten Teil, den Sie mit dem Kästchen "Palette" im Farb-Win-

dow aufrufen, können Sie die RGB-Werte aller verwendeten Farben nach Ihren Vorstellungen ändern. Bitte fügen Sie den folgenden Teil im Listing zwischen den Unterprogrammen 'Farbeinstellung:' und 'Farbabfrage:' ein:

```
PaletteDef:
  IF ZFWahl=0 THEN Neufarbe=Zfarbe ELSE Neufarbe=Ffarbe
  PATTERN ,Voll%
  LINE (0,71)-(240,107),0,bf
  COLOR 1,0
  LOCATE 10,3 : PRINT "R";
  LOCATE 11,3 : PRINT "G";
  LOCATE 12,3 : PRINT "B";
  LINE (24,70)-(218,78),1,b
  LINE (24,80)-(218,88),1,b
  LINE (24,90)-(218,98),1,b
  LINE (222,70)-(238,98),Neufarbe,bf
  Modus=4
  PATTERN ,Muster%
RETURN

RGBDef:
  Test=MOUSE(0)
  x=MOUSE(3) : y=MOUSE(4)
  GOSUB Farbabfrage
  IF ZFWahl=0 THEN Neufarbe=Zfarbe ELSE Neufarbe=Ffarbe
  GOSUB RGBRegler
  GOSUB OKCheck : IF EndOK=1 THEN EndOK=3
  WHILE MOUSE(0)<>0
    x=MOUSE(1) : y=MOUSE(2)
    IF WINDOW(0)=3 AND x>26 AND x<218 AND y>70 AND y<98 THEN
      Farben%(Neufarbe,INT((y-71)/8.7))=INT((x-26)/12)
      GOSUB RGBRegler
    END IF
  WEND
RETURN

RGBRegler:
  PATTERN ,Voll%
  LINE (25+r*12,71)-(37+r*12,77),0,bf
```

```
        LINE (25+g*12,81)-(37+g*12,87),0,bf
        LINE (25+b*12,91)-(37+b*12,97),0,bf
        r=FarbenX(Neufarbe,0)
        g=FarbenX(Neufarbe,1)
        b=FarbenX(Neufarbe,2)
        LINE (25+r*12,71)-(37+r*12,77),1,bf
        LINE (25+g*12,81)-(37+g*12,87),1,bf
        LINE (25+b*12,91)-(37+b*12,97),1,bf
        PALETTE Neufarbe,r/16,g/16,b/16
        LINE (222,70)-(238,98),Neufarbe,bf
        PATTERN ,Muster%
      RETURN
```

Das erste der drei Unterprogramme ist wieder für Vorbereitungen und den Aufbau der Szenerie zuständig: 'PaletteDef:'. Je nach Inhalt von 'ZFWahl' merkt sich die Variable 'Neufarbe' die Farbnummer der Zeichenfarbe oder der Ausmalfarbe. Sie soll verändert werden, solange der Anwender keine andere Farbe auswählt.

Die Schieberegler für Rot, Grün und Blau erscheinen im Farb-Window. Deshalb müssen wir für sie etwas Platz schaffen. Wir verzichten auf die Kästchen der Zeichen- und Ausmalfarbe sowie auf das "Palette"-Kästchen. Die brauchen wir nämlich nicht. Um die drei Kästchen zu löschen, lassen wir den Amiga einfach einen Block in der Hintergrundfarbe (Nummer 0) zeichnen. An die gelöschte Stelle kommen dann die drei Schieberegler und die Buchstaben R, G und B als Beschriftung. Neben die Regler zeichnen wir noch ein Kästchen, das immer die gerade gewählte Farbe haben wird. So erkennen Sie gleich, welche Farbe Sie zur Zeit verändern.

'Modus' ist jetzt Modus 4. Bereits in der 'Maus:'-Routine haben wir ja festgelegt, daß Modus 4 dem Modus 2 untergeordnet ist. Die Einteilung in Modi benötigt das Programm, um festzustellen, was der Anwender mit einem Mausklick erreichen möchte.

Der Teil 'RGBDef:' wertet die Maus-Aktionen aus, solange das Farbdefinitions-Programm aktiv ist. Nachdem die Mauskoordinaten festgestellt wurden, checken wir mit dem Unterprogramm 'Farbabfrage:' ab, ob in eines der Farbkästchen geklickt wurde. Damit können Sie nämlich nachher auswählen, welche Farbe geändert werden soll. Die neue Farbe wird auch gleich der Variablen 'Neufarbe' mitgeteilt.

Als nächstes ist das Unterprogramm 'RGBRegler:' an der Reihe. Es wird je nach den aktuellen Farbwerten die Regler richtig einstellen.

Um zu prüfen, ob vielleicht ins OK-Feld geklickt wurde, leihen wird uns einfach das Programm 'OKCheck:' aus, das ja eigentlich zum 'Farbauswahl:'-Teil gehört. Soll tatsächlich Schluß sein, bekommt 'EndOK' zunächst den Wert 1. Wir machen dann daraus die Zahl 3, und schon ist die Abbruchbedingung für das Farbdefinitions-Programm gegeben. Wenn Sie im Modus 4 ins Feld OK klicken, kehrt das Programm zunächst eine Stufe zurück, das heißt in den Modus 2, die Farbauswahl. Nach nochmaligem Anklicken von OK sind Sie dann wieder im Zeichenmodus. Wenn Ihnen das zu umständlich ist, können Sie auch direkt die Option "Zeichnen" aus dem "Programm"-Menü wählen. Die bringt Sie sofort in den Zeichenmodus zurück.

Die WHILE...WEND-Schleife im 'RBGDef:'-Teil verschiebt die Regler nach Wunsch. Solange die Maustaste gedrückt bleibt, das gewählte Window die Nummer 3 hat und sich der Mauscursor im Bereich der Regler befindet, aktualisieren wir im Datenfeld 'Farben%' ständig einen der Farbanteile (R, G oder B) von 'Neufarbe'. Welcher Farbanteil das ist, hängt von der y-Koordinate des Mauscursors ab. Und wie groß der Farbanteil sein soll, sagt uns die x-Koordinte des Mauscursors. Bevor das IF...THEN und die WHILE...WEND-Schleife beendet werden und das Programm zurückspringt, rufen wir den Programmteil "RGBRegler:" auf. In diesem Teil werden nämlich die Regler anhand der Variablen 'r', 'g' und 'b' an die richtige Position gezeichnet. Zuerst löscht das Unterprogramm die alten Regler, indem es sie in der Hintergrundfarbe zeichnet. Dann liest es 'r', 'g' und 'b' aus dem 'Farben%'-Feld. Das Ergebnis ist immer eine ganze Zahl zwi-

schen 0 und 15. 0 bedeutet "kein Anteil" und 15 "voller Anteil". Die Schieberegler erscheinen wieder, gegebenenfalls an einer veränderten Position. Ein PALETTE-Befehl teilt AmigaBASIC die neue Farbdefinition mit. Schließlich zeichnen wir in der neuen Farbe das Kästchen nach, das die aktuelle Farbe darstellt. Und das war's dann schon. Rücksprung mit RETURN.

Sie werden sehen: Die Bedienung ist wesentlich einfacher als die technischen Erklärungen. Aber das ist ja oft so. (Was glauben Sie, was man alles über Zahnbürsten, Kühlschränke oder Fahrräder schreiben könnte. Allerdings müssen wir zugeben, daß diese Geräte auch selten programmiert werden. Zumindest noch nicht...)

Sie können sich aussuchen, welche Farbe umdefiniert werden soll, indem Sie in eines der Farbfelder klicken. Die Regler für R, G und B verschieben Sie mit der Maus, wie Sie es auch von Preferences oder den kommerziellen Malprogrammen gewohnt sind. Sind alle Farben nach Ihrem Geschmack, können Sie die Farbdefinition mit einem Klick ins OK-Feld wieder verlassen.

Zwischenspiel 4: Von Bits, Bytes und anderen Gemeinheiten

An dieser Stelle müssen wir den Lauf der Dinge mal wieder unterbrechen. Nutzen Sie das ruhig zu einer kleinen Verschnaufpause, bevor Sie weiterlesen.

Im folgenden wird es um die Definition von Füllmustern gehen. Aber vorher müssen Sie ein paar wichtige Grundlagen kennenlernen, die leider ein gewisses Maß an mathematischem Verständnis erfordern. Dieses Zwischenspiel erklärt *Binärzahlen*, *Hexadezimalzahlen* und was man damit alles anstellen kann.

Wir haben schon erklärt, daß ein Computer nur zwei Zustände unterscheiden kann: 0 und 1, Strom an und Strom aus, logisch "wahr" und logisch "falsch". Alle anderen Funktionen werden mehr oder weniger trickreich von diesen beiden Zuständen abgeleitet. Uns interessiert jetzt zum Beispiel, wie sich der Amiga andere Zahlen als 0 und 1 merken kann. Schon mit der Zahl 2

müßte er ganz schön ins Schwitzen kommen: Eine Unterscheidung zwischen "Strom an", "Strom anner" und "Strom am annsten" gibt es schließlich nicht.

Eigentlich kennen Sie den Trick schon. Was haben wir gemacht, als für die Farben ein einzelnes Bit zu wenig wurde? Wir haben uns weitere Bits dazugenommen. Dasselbe macht der Amiga auch, wenn er Zahlen verarbeitet. Mit einem Bit kann er 0 und 1 unterscheiden. Mit zwei Bits kennt er schon 0, 1, 2 und 3. Die ersten 8 Zahlen sehen demnach Amiga-intern so aus:

Zahl	in Bits	Zahl	in Bits
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

Tabelle 6: **Dezimalzahlen und Binärzahlen**

Wie Sie sehen, handelt es sich um dieselben Kombinationen, die wir schon bei den Bit-Ebenen kennengelernt haben. Vielleicht haben Sie, z.B. in der Schule, schon mal vom Zweier-System, Dualsystem oder von *Binärzahlen* gehört. Sehr häufig werden von Lehrern hier irgendwelche armen Außerirdischen herangezogen und schrecklich verstümmelt: "Stellt euch vor, es gäbe auf einem anderen Planeten Lebewesen, die nur zwei Finger hätten..." und so weiter. Dabei wissen wir, spätestens seit E.T., daß da weitaus mehr Finger zu finden sind. Na egal - auf jeden Fall geht es darum, uns klarzumachen, daß man auch nur mit den Ziffern 0 und 1 alle Zahlen unseres sogenannten Dezimalsystems darstellen kann. Der gewitzte Schüler, der nun nachhakt und wissen will, wozu man das machen soll, wenn man doch zehn gesunde und kräftige Ziffern zur Verfügung hat, wird mit dem schon erwähnten außerirdischen Besucher zum Schweigen gebracht, der halt nun mal nur das Dualsystem versteht. Weil man aber nie weiß, wer uns besuchen kommt, muß man sich halt damit beschäftigen.

Was im Mathematik-Unterricht nach einer spitzfindigen Spielerei aussieht, findet bei Computern eine sinnvolle Anwendung. Unser normales Zahlensystem, das Dezimalsystem, benutzt 10 Ziffern (0, 1, 2, ..., 9). Wenn die zehn Ziffern nicht mehr ausreichen, wird eine neue Stelle eröffnet: Für 9 reicht noch eine Stelle aus, für 10 brauchen wir zwei. Die Werte der einzelnen Stellen sind immer Potenzen der Grundzahl ($1=10^0$, $10=10^1$, $100=10^2$ usw.).

Macht man sich Gedanken darüber, warum es ausgerechnet die Zahl 10 sein mußte, muß man ehrlich zugeben, daß es eigentlich keinen besonderen Grund dafür gibt. Wahrscheinlich liegt es daran, daß der Mensch zum Rechnen schon immer Hilfsmittel brauchte. Und da wir nun mal zehn Finger haben - im Gegensatz zu manchen Außerirdischen, die offensichtlich nur die Bekanntschaft von Mathematiklehrern suchen...

Man kann Zahlensysteme aber auch auf jeder anderen Grundzahl aufbauen.

Computer kennen zwei Ziffern (0 und 1), also rechnen sie im Zweier-System. Bereits für die Zahl 2 muß eine neue Stelle aufgemacht werden. Dasselbe Spielchen folgt bei allen weiteren Potenzen von 2: $2^0=1$, $2^1=2$, $2^2=4$, $2^3=8$, $2^4=16$ usw. Für jede dieser Zahlen benötigt das Binärsystem eine neue Stelle. Das hat den Nachteil, daß schon kleine Zahlen verhältnismäßig lang werden. Davon abgesehen, funktioniert das Ganze aber genauso gut wie das Dezimalsystem auch. Wenn Sie den Wert einer Binärzahl ausrechnen wollen, müssen Sie einfach die Werte der besetzten Stellen aufaddieren.

Nehmen wir z.B. die Binärzahl 10011011. So sieht die Besetzung der einzelnen Stellen aus:

Wert:	128	64	32	16	8	4	2	1
	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Inhalt:	1	0	0	1	1	0	1	1

Die Binärzahl 10011011 entspricht dem dezimalen Wert 128+16+8+2+1, also 155.

Aus technischen Gründen hat es sich eingebürgert, acht Bits zu einer Einheit zusammenzufassen. Das nennt man dann *Byte*. Ein Byte hat 8 Bits, kann also höchstens den Wert 255 darstellen. (Wenn Sie das nachprüfen wollen, rechnen Sie doch einfach die Werte der Stellen in dem obigen Beispiel zusammen.) Eine Speicherzelle im Amiga kann genau ein Byte speichern. Diese technische Gegebenheit geht auf die Zeit zurück, als es nur 8-Bit-Prozessoren gab. Der Hauptprozessor des Amiga, der 68000, kann allerdings auch 16 Bits auf einen Schlag verarbeiten.

Einen 16-Bit-Wert nennt der Fachmann "Wort". Über den Sinn dieser Bezeichnung läßt sich streiten, es hat auf jeden Fall wenig damit zu tun, was Sie und wir normalerweise unter einem Wort verstehen. Ein "Wort" besteht aus zwei Bytes und wird in zwei hintereinanderliegenden Speicherzellen gespeichert. Mit 16 Bits kann man Zahlen von 0 bis 65535 darstellen.

Auch 32-Bit-Werte können vom 68000-Prozessor verarbeitet werden. Die heißen dann "Langwort" und liegen zwischen 0 und 4294967294. Eine ganz schön große Zahl, es sind immerhin über 4 Milliarden. Diese Zahl brauchen Sie sich natürlich nicht zu merken.

Wir werden auch "Wort" und "Langwort" in diesem Buch nicht mehr gebrauchen. Wir bringen soetwas einfach nicht mehr als einmal aus der Feder. Sie sollten die beiden Ausdrücke nur kennengelernt haben, falls Sie sie mal irgendwo anders lesen.

Es kann vorkommen, daß nur 15 bzw. 31 Bit zur Speicherung einer Zahl verwendet werden und das höchstwertige Bit zur Speicherung des Vorzeichens dient. Eine 0 auf dieser Stelle bedeutet dann positiv, eine 1 steht für negativ. Auch AmigaBASIC handhabt das so: 16-Bit-Zahlen liegen im Bereich von -32768 bis +32767, 32-Bit-Zahlen zwischen -2147483647 und +2147483646.

Die Zahl 2147483646 beweist, daß auch Dezimalzahlen sehr lang werden können, und dabei nicht gerade übersichtlich bleiben. Das ist einer der Gründe, warum sich Programmierer, nachdem sie die Binärzahlen schon mal verstanden hatten, noch ein anderes Zahlensystem zugelegt haben. Die Rede ist von *Hexadezimalzahlen*. Diese Zahlen haben die Basis 16. Als Ziffern mit dem Wert 10 bis 15 werden die Buchstaben A bis F benutzt. Ein Vergleich der drei Zahlensysteme zeigt Ihnen die Unterschiede:

Dez.	Bin.	Hex.	Dez.	Bin.	Hex.
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Tabelle 7: **Die drei Zahlensysteme im Vergleich**

Das Hexadezimalsystem bietet im Zusammenhang mit Computern mehrere Vorteile: Erstens, das wissen Sie jetzt schon, können große Zahlen kürzer geschrieben werden. Der Gewinn von ein oder zwei Stellen ist zwar nicht immens, aber immerhin... Zweitens ist 16 eine Zweier-Potenz. Das heißt, daß Binärzahlen und Hexadezimalzahlen relativ einfach ineinander umgerechnet werden können. Immer vier Stellen einer Binärzahl entsprechen einer Hex-Ziffer. Das sehen Sie auch in der Abbildung oben.

Wo erkennen Sie schneller, wieviele Bits an sind? Bei 65535 oder bei FFFF? Sie sehen, das Hex-System ist gar nicht so schlecht...

Wenn Sie in AmigaBASIC Hexadezimalzahlen verwenden, müssen sie besonders gekennzeichnet werden. Das sieht dann so aus:

```
? &h7fff
```

Weil wir gerade dabei sind: AmigaBASIC kennt noch ein Zahlensystem, nämlich Oktalzahlen. Sie haben die Basis 8 und Ziffern zwischen 0 und 7. Ein Beispiel:

```
? &o7777
```

Oktalzahlen brauchen Sie allerdings sehr selten. Hexadezimalzahlen und Binärzahlen hingegen kommen recht häufig vor und sind sehr hilfreich. AmigaBASIC kennt leider keine Möglichkeit, Binärzahlen direkt darzustellen. Deshalb benutzen wir in Programmen in erster Linie Dezimalzahlen und Hexadezimalzahlen. Stellt sich die Frage, was Sie mit diesen Zahlensystemen eigentlich anfangen sollen. Sie können ganz normal damit rechnen:

```
? &hf + &ha02
```

entspricht 15+2562 und ergibt 2577. Die Schreibweise einer Zahl interessiert AmigaBASIC nicht, nur der Wert zählt. Sie können alle Zahlentypen auch gemischt verwenden:

```
? 15 + &ha02
```

Aber das war noch lange nicht alles. Es gibt noch die sogenannten logischen Verknüpfungen. Das klingt ja schon wieder sehr mathematisch, ist aber halb so schlimm. (...sagte Mr. Spock und lachte mit bewegungslosem Gesicht.) Die Befehle AND und OR kennen Sie schon. Wir haben sie bisher hauptsächlich in IF...THEN-Befehlen benötigt, um mehrere Bedingungen zu verbinden. AND bedeutet "sowohl als auch": Beide Bedingungen müssen "wahr" sein, damit das Ergebnis "wahr" ist. OR bedeutet "oder": Eine "wahre" Bedingung reicht aus, damit das Ergebnis "wahr" ist.

Sie haben auch schon gehört, daß AmigaBASIC den Wahrheitsgehalt einer Aussage durch 0 und -1 ausdrückt. Wir bringen nun ein bißchen System in die ganze Angelegenheit. Befehle wie

AND und OR werden benutzt, um Zahlen miteinander zu verknüpfen. Dazu werden immer einzelne Bits miteinander verglichen. Fangen wir mal mit AND an. Wenn Sie

```
? 59 and 93
```

eingeben, liefert der Amiga als Ergebnis die Zahl 25. Wie kommt's? Ganz einfach:

```
      0 0 1 1 1 0 1 1      (59)
AND   0 1 0 1 1 1 0 1      (93)
      _______________
      0 0 0 1 1 0 0 1      (25)
```

Im Ergebnis wird dann ein Bit gesetzt, wenn in beiden verknüpften Zahlen das Bit an der jeweiligen Stelle gesetzt ist. Das folgende Schema zeigt die möglichen AND-Verknüpfungen und ihr Ergebnis. So eine Übersicht über die Auswirkungen einer Verknüpfung nennt man übrigens "Wahrheitstabelle". Die Wege der Sprachschöpfer sind eben unergründlich und verworren.

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Beim Verknüpfen von Zahlen (die Sie in beliebiger Schreibweise angeben können) vergleicht AmigaBASIC Bit für Bit und setzt eine Ergebniszahl zusammen.

Die nächste Verknüpfung ist OR. Hier wird dann ein Bit im Ergebnis gesetzt, wenn ein Bit oder beide Bits gesetzt sind. Wieder ein Beispiel:

```
? 77 OR 132
```

ergibt 205.

	0 1 0 0 1 1 0 1	(77)	0 OR 0 = 0
OR	1 0 0 0 0 1 0 0	(132)	0 OR 1 = 1
	———————		1 OR 0 = 1
	1 1 0 0 1 1 0 1	(205)	1 OR 1 = 1

Weil's so schön war, gibt es noch mehr Verknüpfungen. Zum Beipiel XOR (gesprochen Ex-Or, kommt von Exklusiv-Oder). Nicht daß diese Funktion besonders exklusiv wäre, exklusiv bedeutet vielmehr "ausschließlich": *Entweder* ein Bit *oder* das andere muß gesetzt sein - sind beide gesetzt, bleibt das Ergebnis-Bit aus:

```
? 90 XOR 213
```

ergibt 143.

	0 1 0 1 1 0 1 0	(90)	0 XOR 0 = 0
XOR	1 1 0 1 0 1 0 1	(213)	0 XOR 1 = 1
	———————		1 XOR 0 = 1
	1 0 0 0 1 1 1 1	(143)	1 XOR 1 = 0

Die nächste Verknüpfung heißt EQV, steht für "Äquivalenz", auf deutsch: Gleichartigkeit. Das Ergebnisbit wird gesetzt, wenn die beiden verglichenen Bits gleich sind. Diese Funktion wird relativ selten gebraucht. Sie ist auch nicht ganz so einfach wie die bisherigen Funktionen, da hier die Vorzeichen der Zahlen immer mitberücksichtigt werden:

```
? 106 EQV -42
```

ergibt 67.

	0 0 0 1 1 0 1 0 1 0	(106)	0 EQV 0 = 1
EQV	1 1 1 1 0 1 0 1 1 0	(-42)	0 EQV 1 = 0
	—————————		1 EQV 0 = 0
	0 0 0 1 0 0 0 0 1 1	(67)	1 EQV 1 = 1

Die nächste Verknüpfung mutet etwas exotisch an: IMP steht für den jedem sofort verständlichen mathematischen Ausdruck "Implikation". "Implizieren" bedeutet "einbeziehen". Beim Vergleich der Werte wird das zweite Bit in das erste einbezogen. Zugegeben, das ist auch noch keine klare Definition. Doch Sie werden diese Funktion in der Praxis so selten brauchen, daß wir es nicht ausführlicher machen wollen. Schauen Sie bei Bedarf einfach die Wahrheitstabelle an.

```
? 370 IMP -474
```

ergibt -337.

	0 1 0 1 1 1 0 0 1 0	(370)	0 IMP 0 = 1
IMP	1 0 0 0 1 0 0 1 1 0	(-474)	0 IMP 1 = 1
	———————————		1 IMP 0 = 0
	1 0 1 0 1 0 1 1 1 1	(-337)	1 IMP 1 = 1

Haben Sie keine Angst, daß nun noch kompliziertere Operationen folgen. Der nächste und letzte Befehl heißt NOT. NOT kehrt einen logischen Wert um. Aus 0 wird -1 und aus -1 wird 0. Logisch "wahr" wird ja bei Vergleichen durch die Zahl -1 ausgedrückt. Das ist leider nicht ganz konsequent (bisher war ja 1 das "Gegenteil" von 0), aber nicht zu ändern. NOT rechnet die Zahlen nach folgender Formel um:

Neuer Wert = -Alter Wert-1

Damit kann 0 in -1 und -1 in 0 umgerechnet werden. Bei allen anderen Zahlen macht die Anwendung von NOT nicht viel Sinn. (Wieso sollte NOT 2 ausgerechnet -3 sein?) NOT wird daher fast ausschließlich bei IF...THEN oder ähnlichen Vergleichen verwendet. Die beiden folgenden BASIC-Zeilen bewirken dasselbe:

```
IF NOT (a$="Hallo") THEN ...
```

und

```
IF a$<>"Hallo" THEN ...
```

Und schon haben Sie's geschafft. Nun wissen Sie alles über Bits, Bytes, Zahlensysteme und logische Verknüpfungen. Dieses Wissen wird für Sie in diesem Buch und erst recht beim späteren Programmieren sehr hilfreich sein. Das werden Sie schon im nächsten Kapitel merken. Nebenbei bemerkt: Logische Verknüpfungen sind sicher nicht einfach, aber glauben Sie uns, Teppiche zu knüpfen, ist viel schwerer. Lassen Sie sich das ein Trost sein.

Weil dieses Zwischenspiel etwas trocken war, wollen wir Sie auch gleich entschädigen: Wir kommen zu den lang erwarteten Füllmustern in AmigaBASIC und bringen so das Malprogramm zu einem guten Ende.

2.10 Hier ein Pünktchen, da ein Pünktchen - Füllmusterdefinition

Wie so vieles haben wir auch die Füllmuster dem Blitter zu verdanken: Er ist nicht nur in der Lage, Flächen zu kopieren und auszumalen, er kann dabei auch noch recht komplexe Verknüpfungen von Flächen und Mustern durchführen. Dazu benutzt er dieselben logischen Verknüpfungen, die wir im letzten Zwischenspiel kennengelernt haben.

Es ist ein leichtes für ihn, einen Block oder ein Vieleck (das mit AREA erzeugt wird) nicht nur mit einer einfarbigen vollen Fläche zu füllen, sondern zum Ausmalen beispielsweise Herzchen, Sternchen oder kleine "Amiga-A's" zu verwenden. Die Muster können Sie selbst entwerfen. Alles was der Blitter braucht, sind Daten, die ihm Auskunft darüber geben, wie das gewünschte Muster aussieht.

Wie unterstützt BASIC das Arbeiten mit Mustern? Dazu gibt es den PATTERN-Befehl, den Sie vom bisherigen Abtippen ja schon zur Genüge kennen. "Pattern" heißt auf Deutsch "Muster". Hinter PATTERN werden die Daten angegeben, die die gewünschten Muster beschreiben. Das sieht so aus:

PATTERN (Wert für Linien), (Datenfeld für Flächen)

Sie können also an erster Stelle ein Muster festlegen, das zum Zeichnen von Linien benutzt werden soll. Dieses Muster wird durch eine 16-Bit-Zahl beschrieben: Sie haben 16 horizontal nebeneinanderliegende Punkte zur Verfügung, um Ihr Muster anzugeben. Wollen Sie die Linien z.B. in einem "Punkt an/Punkt aus"-Muster zeichnen, sehen die Bits und ihre Zahlenwerte so aus:

Bits: 01010101 01010101
Dez.: 21845
Hex.: &H5555

Wir haben gleich den dezimalen und den hexadezimalen Wert ausgerechnet. Wenn Sie Im BASIC-Window

```
pattern &h5555
```

eingeben, wird das oben durch Bits dargestellte Muster für Linien übernommen. Ein Bit mit dem Wert 1 entspricht einem sichtbaren Punkt, ein 0-Bit einem unsichtbaren. Probieren Sie, damit im BASIC-Window ein paar Linien zu zeichnen:

```
for x=1 to 1000 : line (635*rnd,185*rnd)-(635*rnd,185*rnd) : next
```

Sie müssen schon genau hinsehen, um die einzelnen Pixels zu erkennen. Versuchen Sie dasselbe vielleicht besser mit einem anderen Muster:

```
pattern &hcccc
```

Jetzt ist das Muster der Linien deutlich sichtbar. Sie geben mit PATTERN ein Bitmuster an und zeichnen mit den gewohnten Grafikbefehlen - um mehr müssen Sie sich nicht kümmern.

Übrigens haben Sie gerade noch etwas Neues gelernt: Ein Füllmuster bzw. ein Linienmuster bleibt so lange aktiv, bis es durch ein anderes ersetzt wird. Standardmäßig verwendet der Amiga das Muster "alle Punkte an" (das entspricht der Hexadezimalzahl

&HFFFF, falls Sie jetzt oder später irgendwann einmal den Urzustand wiederherstellen wollen).

Vorsicht beim Ausmalen! Die unsichtbaren Stellen in den Linien sind wirklich nicht da. Wir verwenden zum Zeichnen keine durchgehende Linie mehr, sondern nur eine Ansammlung von Punkten. Und da läuft die Farbe auf jeden Fall durch. Deshalb haben wir im Malprogramm auch auf die Möglichkeit verzichtet, Muster für die Linien zu definieren. Denn Linien dienen ja fast immer als Begrenzung für farbige Flächen. Dafür können Sie im Malprogramm Füllmuster für Flächen angeben. Das ist die zweite Möglichkeit des PATTERN-Befehls. Nur wird für Flächen keine einzelne Zahl, sondern der Name eines Datenfelds angeben, das Integer-Zahlen (ganze Zahlen) enthält:

PATTERN ,(Feldname)

Jetzt wissen Sie auch, warum im Listing beim PATTERN-Befehl vor den Feldern 'Voll%' und 'Muster%' immer ein Komma steht: Wir haben den Wert für die Linien-Definition einfach weggelassen.

Welche Werte müssen in dem Datenfeld stehen? Sie wissen schon, daß es Integer-Zahlen sein sollen. Das ist deshalb sinnvoll, weil Nachkommastellen bei der Darstellung in Bits sowieso nur sehr umständlich zu erreichen sind. Für die Füllmuster, mit denen geschlossene Flächen ausgemalt sowie Areas und Blocks gezeichnet werden, haben Sie in der Breite wieder 16 Punkte Platz. Die Muster können aber beliebig hoch werden. Einzige Regel: Die Höhe des Musters in Bits muß eine Potenz von 2 sein. (Also 2, 4, 8, 16 etc.) Wir verwenden im Malprogramm 8 Pixel-Reihen, sprich 16 * 8 Punkte pro Muster. Die Flächen können natürlich im Bild viel größer werden, in der Fläche wiederholt sich das Muster. Im Malprogramm können Sie immer eines aus 9 verschiedenen Mustern auswählen - und alle Muster beliebig verändern.

Damit wären wir soweit, daß Sie erst einmal die Programmzeilen eingeben können, falls Sie das überhaupt tun.

```
Mustereditor:
  MOUSE OFF : MENU OFF
  EndOK=0
  WINDOW 4,"Füllmuster",(54,30)-(300,130),18,1
  LINE (0,0)-(132,66),3,b

  FOR x=0 TO 2
    FOR y=0 TO 2
      FOR i=0 TO 7 : Muster%(i)=Allemuster%(y*3+x,i) : NEXT
      PATTERN ,Muster%
      LINE (144+x*34,y*25)-(175+x*34,23+y*25),1,bf
    NEXT y
  NEXT x
  GOSUB Markmuster

  LINE (5,68)-(65,82),1,b
  LOCATE 10,2 : PRINT "Lösch";
  LINE (75,68)-(135,82),1,b
  LOCATE 10,11 : PRINT "Inv.";
  LINE (5,85)-(65,100),1,b
  LOCATE 12,2 : PRINT "Load";
  LINE (75,85)-(135,100),1,b
  LOCATE 12,11 : PRINT "Save";
  LINE (162,77)-(222,92),1,b
  LOCATE 11,24 : PRINT "OK";

  FOR i=0 TO 7 : Muster%(i)=Allemuster%(Musternr,i) : NEXT i
  GOSUB Zeigmuster

  MENU ON : MOUSE ON
RETURN

Musterdef:
  Test=MOUSE(0)
  x=MOUSE(3) : y=MOUSE(4)
  IF WINDOW(0)=4 AND x<132 AND y<66 THEN
    px=INT(x/8.25) : py=INT(y/8.25)
    Bit=Muster%(py) AND 2^(15-px)
    IF Bit=0 THEN
      Muster=Muster%(py) OR 2^(15-px)
```

```
    ELSE
      Muster=Muster%(py) AND (65535-2^(15-px))
    END IF
    IF Muster>32767 THEN Muster=Muster-65536
    Muster%(py)=Muster
    PATTERN ,Voll%
    LINE (px*8+4,py*8+2)-(px*8+9,py*8+8),-(Bit=0),bf
    PATTERN ,Muster%
    y1=INT(Musternr/3) : x1=Musternr-y1*3
    LINE (144+x1*34,y1*25)-(175+x1*34,23+y1*25),1,bf
    FOR i=0 TO 7 : Allemuster%(Musternr,i)=Muster%(i) : NEXT
    RETURN
  END IF

  IF WINDOW(0)=4 AND x>142 AND x<244 AND y<75 THEN
    px=INT((x-143)/34) : py=INT(y/25)
    IF px+py*3=Musternr THEN RETURN
    Musternr=px+py*3
    FOR i=0 TO 7 : Muster%(i)=Allemuster%(Musternr,i) : NEXT
    GOSUB Markmuster
    GOSUB Zeigmuster
    PATTERN ,Muster%
    RETURN
  END IF

  IF WINDOW(0)=4 AND x<222 AND x>162 AND y<93 AND y>76 THEN
    PATTERN ,Voll%
    PAINT (164,78),2,1
    PATTERN ,Muster%
    EndOK=2 : RETURN
  END IF

  IF WINDOW(0)=4 AND x<135 AND y>68 AND y<100 THEN
    PATTERN ,Voll%
    IF x<66 AND x>4 AND y<82 THEN
      PAINT (6,69),2,1
      LINE (1,1)-(131,65),0,bf
      FOR i=0 TO 7 : Allemuster%(Musternr,i)=0 : Muster%(i)=0 : NEXT
      PAINT (6,69),0,1
      PATTERN ,Muster%
```

```
      y1=INT(Musternr/3) : x1=Musternr-y1*3
      LINE (144+x1*34,y1*25)-(175+x1*34,23+y1*25),1,bf
    END IF
    IF x<136 AND x>74 AND y<82 THEN
      PAINT (76,69),2,1
      FOR i=0 TO 7
        Muster%(i)=Muster%(i) XOR &HFFFF
        Allemuster%(Musternr,i)=Muster%(i)
      NEXT i
      GOSUB Zeigmuster
      PAINT (76,69),0,1
      PATTERN ,Muster%
      y1=INT(Musternr/3) : x1=Musternr-y1*3
      LINE (144+x1*34,y1*25)-(175+x1*34,23+y1*25),1,bf
    END IF
    IF x<66 AND x>4 AND y>84 THEN GOSUB Musterladen
    IF x<135 AND x>75 AND y>84 THEN GOSUB Musterspeichern
  END IF
RETURN

Markmuster:
  y1=INT(Altmuster/3) : x1=Altmuster-y1*3
  LINE (143+x1*34,y1*25-1)-(176+x1*34,24+y1*25),0,b
  y1=INT(Musternr/3) : x1=Musternr-y1*3
  LINE (143+x1*34,y1*25-1)-(176+x1*34,24+y1*25),3,b
  Altmuster=x1+y1*3
RETURN

Zeigmuster:
  MOUSE OFF : MENU OFF
  PATTERN ,Voll%
  LINE (1,1)-(131,65),0,bf
  FOR y=0 TO 7
    FOR x=0 TO 15
      Bit=Muster%(y) AND 2^(15-x)
      IF Bit<>0 THEN LINE (x*8+4,y*8+2)-(x*8+9,y*8+8),1,bf
    NEXT x
  NEXT y
```

```
    PATTERN ,Muster%
    MOUSE ON : MENU ON
  RETURN

  Musterladen:
    MOUSE OFF : MENU OFF
    PAINT (6,86),2,1
    GOSUB Eingabename
    IF Nam$="" THEN EndeMLaden
    OPEN Nam$ FOR INPUT AS 1
      FOR x=0 TO 8
        FOR y=0 TO 7
          Allemuster%(x,y)=CVI(INPUT$(2,1))
        NEXT y
      NEXT x
    CLOSE 1

  EndeMLaden:
    WINDOW CLOSE 5 : WINDOW 4
    PAINT (6,86),0,1
    MOUSE ON : MENU ON
    FOR x=0 TO 8
      FOR y=0 TO 7
        Muster%(y)=Allemuster%(x,y)
        PATTERN ,Muster%
        y1=INT(x/3) : x1=x-y1*3
        LINE (144+x1*34,y1*25)-(175+x1*34,23+y1*25),1,bf
      NEXT y
    NEXT x
    FOR i=0 TO 7 : Muster%(i)=Allemuster%(Musternr,i) : NEXT
    GOSUB Zeigmuster
  RETURN

  Musterspeichern:
    MOUSE OFF : MENU OFF
    PAINT (78,86),2,1
    GOSUB Eingabename
    IF Nam$="" THEN EndeMSpeichern
    OPEN Nam$ FOR OUTPUT AS 1
      FOR x=0 TO 8
```

```
      FOR y=0 TO 7
        PRINT #1,MKI$(Allemuster%(x,y));
      NEXT y
    NEXT x
  CLOSE 1

EndeMSpeichern:
  WINDOW CLOSE 5 : WINDOW 4
  PAINT (78,86),0,1
  MOUSE ON : MENU ON
RETURN

Musteraus:
  MENU 2,0,1 : Modus=1
  WINDOW CLOSE 4
  WINDOW OUTPUT 2
  PATTERN ,Muster%
RETURN

Eingabename:
  Altname$=Nam$
  WINDOW 5,"Bitte Dateinamen eingeben:",(0,80)-(311,88),0,1
  CLS
  LINE INPUT Nam$
  IF Nam$= "=" OR Nam$="*" THEN Nam$=Altname$
RETURN
```

Sie sehen schon: Es gibt immer weniger BASIC-Befehle im Grafik-Bereich, die Sie noch nicht kennen.

Wie wir es vom Farb-Unterprogramm schon kennen, baut der erste Teil das Window und seinen Inhalt auf. Das zuständige Unterprogramm heißt 'Mustereditor:'. Während des Aufbaus wird Event Trapping abgeschaltet. Im neuen Window erscheinen zuerst die neun zur Verfügung stehenden Füllmuster. Sie sind ja im Feld 'Allemuster%' gespeichert und werden von den verschachtelten Schleifen auf den Bildschirm gebracht. Das %-Zeichen hinter den Feldnamen sorgt dafür, daß nur Integer-Zahlen gespeichert werden. Sie kennen das schon von normalen

Variablen, bei Feldern funktioniert die Kennzeichnung genauso. Aus dem Feld 'Allemuster%' werden die 8 Muster-Daten zunächst ins Feld 'Muster%' kopiert. In 'Muster%' steht immer das aktuelle, gerade gewählte Muster. Pro Muster zeichnet das Unterprogramm ein kleines, gefülltes Quadrat. Da wir am Programmanfang aus den DATA-Zeilen in alle Felder eine Grundbelegung eingelesen haben, stehen bereits neun fertige Muster zur Verfügung.

Als nächstes wird das Unterprogramm 'Markmuster:' aufgerufen. Es zeichnet einen orangen Rahmen um das aktuelle Füllmuster, damit Sie die Auswahl immer sofort erkennen. Der dann folgende Teil erzeugt die Kästchen für "Lösch" (Muster löschen), "Inv." (Muster invertieren, umkehren), "Load" (Musterbelegung von Diskette laden), "Save" (Muster auf Diskette abspeichern) und "OK". Um eine dieser Funktionen aufzurufen, muß der Anwender in das zugehörige Kästchen klicken.

Zu guter Letzt rufen wir noch 'Zeigmuster:' auf: Dieses Unterprogramm stellt die Vergrößerung des aktuellen Musters in der Arbeitsmatrix dar. Die Arbeitsmatrix ist das große Kästchen links neben den Füllmustern. Hier können Sie einzelne Punkte an- und ausklicken und auf diese Weise Muster verändern oder neu definieren. Am Schluß müssen wir Event Trapping wieder aktivieren und mit RETURN zurückspringen.

So sieht das "Füllmuster"-Window aus, das im gerade besprochenen Teil aufgebaut wird:

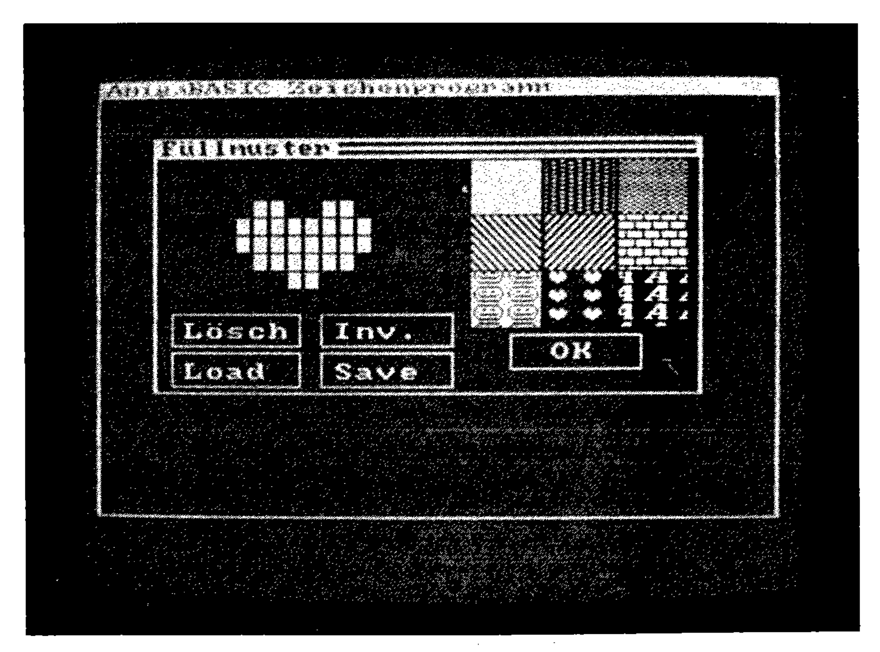


Bild 11: Das "Füllmuster"-Window aus dem Malprogramm

Mit dem Bild können Sie vergleichen, ob in Ihrer Version alles stimmt. Wenn sich das eine oder andere Ihrer Muster von unseren unterscheidet, obwohl Sie nichts verändert haben, überprüfen Sie bitte zunächst die DATA-Zeilen am Programmanfang.

Der Teil 'Musterdef:' ist nun an der Reihe. Dieses Unterprogramm reagiert auf die Maus, solange Modus 3 (die Muster-Definition) aktiv ist. Wir stellen zuerst die Koordinaten der Maus fest und speichern sie in den Variablen 'x' und 'y'. Dann wird Schritt für Schritt abgeprüft, in welchem Feld der Klick erfolgte und was demnach vom Programm zu tun ist.

Es beginnt mit der Arbeitsmatrix. Hier wird, wie schon erwähnt, jeder Punkt des Musters stark vergrößert durch ein Kästchen dargestellt. Sie können durch Klicken mit der Maus einzelne Punkte an- oder ausschalten. Falls das gewählte Window die Nummer 4 ist und mit der Maus innerhalb der Grenzen der Arbeitsmatrix geklickt wurde, soll der angeklickte Punkt berechnet werden. In den Variablen 'px' und 'py' rechnen wir die Mauskoordinaten in eine Spalte und eine Zeile bzw. Reihe um, die die Lage des Punktes angibt, auf den Sie geklickt haben. Die Variable 'Bit' isoliert das zum Punkt gehörende Bit aus dem Feld 'Muster%'. Dazu benutzen wir eine AND-Verknüpfung: Nur wenn in der Ausgangszahl ('Muster%(py)') dasselbe Bit wie in der Vergleichszahl (2^(15-px)) gesetzt ist, wird das Bit auch im Ergebnis gesetzt. Anderenfalls kommt der Wert 0 heraus. Hat 'Bit' den Wert 0, wird der angeklickte Punkt im Muster gesetzt.

Ist das Ergebnis ein anderer Wert als 0, soll der Punkt gelöscht werden. Dazu führen wir AND mit einer Zahl aus, in der alle Bits gesetzt sind bis auf das eine, das dem gewählten Pixel entspricht (Die Formel dazu lautet: 65536-2^(15-px)). Das Resultat: Alle Bits bleiben unverändert, nur das gewünschte Bit wird gelöscht.

Die Variable 'Muster' erhält einen neuen Wert, der zurück ins 'Muster%'-Feld geschieben wird. Um die Zahl in das Integer-Feld zurückschreiben zu können, müssen wir noch etwas beachten: Unser Ergebnis hat zwar 16 Bit, ist aber auf jeden Fall positiv. Um die entsprechende Integer-Zahl zu erhalten, müssen wir von Zahlen, die größer als 32767 sind, den Wert 65535 abziehen. So kommen wir auf den Negativ-Wert. Der wird dann schließlich ins Feld 'Muster%' zurückgeschrieben. Wundern Sie sich nicht, daß AmigaBASIC hinter der Zahl 65535 im LIST-Window beharrlich ein &-Zeichen anhängt. Das macht es mit allen Zahlen, die außerhalb des Bereichs von -32768 bis +32767 liegen.

Damit ist der rechnerische Teil der Muster-Definition schon erledigt. Aber der Anwender will ja auch eine Reaktion sehen. Wobei die Betonung auf "sehen" liegt. Je nach dem vorherigen Zustand des Punkts wird er in der Arbeitsmatrix gesetzt oder

gelöscht. Die Formel -(Bit=0) kehrt den Wert eines Bits um. Das hängt mal wieder mit der Verwaltung von wahren und falschen Aussagen zusammen: Hat 'Bit' den Wert 0, ist das Ergebnis 1 und umgekehrt.

Wir verwenden das Ergebnis der Formel als Zeichenfarbe: Mit der Farbe 0 (Hintergrund, also unsichtbar) oder 1 (erste Zeichenfarbe, also sichtbar) zeichnet ein LINE-Befehl ein ausgemaltes Kästchen. Der Zustand des angeklickten Punktes hat sich dann genau umgekehrt: War er vorher an, ist er jetzt aus; war er aus, wird er gesetzt. Vorher muß das Füllmuster auf 'Voll%' zurückgeschaltet werden, denn wir wollen ja ein gefülles Kästchen und kein gemustertes. Jetzt verstehen Sie sicher, warum wir im ganzen Programm vor Blockfill-Operationen immer erst PATTERN ,Voll% verwenden. In 'Voll%' haben wir ja ganz am Anfang ein Füllmuster abgelegt, bei dem alle Punkte gesetzt sind. Wenn wir außerhalb des Bilds, also zum Aufbau von Windows, volle Flächen verwenden wollen, sind keine Muster gefragt.

Damit der Anwender sieht, wie sich das Muster in Originalgröße macht, bringen wir in dem Kästchen, in dem das alte Muster dargestellt wurde, die neue Version des Musters auf den Bildschirm. Die Tatsache, daß der Inhalt des Felds 'Muster%' geändert wurde, ändert ja nichts an Mustern, die sich schon auf dem Bildschirm befinden. Wir müssen das Kästchen mit LINE und Blockfill neu zeichnen.

Dieses Prinzip können Sie sich auch beim Malen Ihrer Bilder zunutze machen: Wenn Sie ein neues Muster auswählen oder einen neuen Mustersatz von Diskette laden, bleiben die Muster im Bild trotzdem unverändert. Der Grund ist, daß der Blitter beim Zeichnen das jeweilige Muster in den Speicher kopiert, wo das Bild liegt. Sie könnten dasselbe Ergebnis im Bild auch dadurch erreichen, daß Sie jeden Punkt des Musters von Hand setzen, nur würde das natürlich sehr viel länger dauern und sehr mühsam sein.

Das Unterprogramm 'Musterdef:' legt den Inhalt des Felds 'Muster%' noch an der entsprechenden Stelle von 'Allemuster%' ab, dann hat es seine Schuldigkeit getan und kann beendet werden.

Der nächste Teil ist dafür zuständig, daß der Anwender das aktuelle Muster wechseln kann. Er kann es dann zum Zeichnen verwenden oder es auch umdefinieren. Klicken Sie einfach in das Feld des Musters, das Sie benutzen oder ändern möchten. Es wird durch einen orangen Rahmen gekennzeichnet, und sein Inhalt wird in die Arbeitsmatrix übernommen. Die Variable 'Musternr' enthält die Nummer des aktuellen Musters. Erfolgte der Klick auf dem Muster, das sowieso gerade gewählt ist, springt das Programm gleich mit RETURN zurück, um Zeit zu sparen. Die acht Daten werden aus dem Feld 'Allemuster%' ins Feld 'Muster%' kopiert. Dann rufen wir 'Markmuster:' und 'Zeigmuster:' auf. Nochmal, was dort passiert: Oranges Kästchen um aktuelles Muster zeichnen und Muster punktweise in die Arbeitsmatrix übernehmen. Letzteres nimmt die meiste Zeit in Anspruch, bitte gedulden Sie sich einen Augenblick. Mit PATTERN wird das neue Muster fürs Zeichnen aktiviert.

Der nächste Teil führt die Auswertung eines Klicks ins OK-Feld durch. Das ist besonders einfach: OK-Feld ausmalen, 'EndOK'=2 und zurück. Um den Rest kümmert sich der aufrufende Programmteil 'Maus:'.

Jetzt fehlen uns in diesem Programmteil nur noch die vier Kästchen für "Lösch", "Inv.", "Load" und "Save". Die äußere IF...THEN-Bedingung fragt ab, ob die Koordinate, an die mit der Maus geklickt wurde, innerhalb des Bereichs liegt, wo sich die vier Felder befinden. Falls nein, kann gleich weitergemacht werden, wodurch das Programm wieder Zeit spart. Die inneren IF...THEN-Abfragen überprüfen ein Kästchen nach dem anderen.

Es geht los mit dem "Lösch"-Kästchen: Das angeklickte Kästchen wird zur Bestätigung des Klicks ausgemalt. Wie immer natürlich erst, nachdem wir das 'Voll%'-Muster aktiviert haben. Innerhalb der Arbeitsmatrix zeichnet der LINE-Befehl ein Kästchen in Hintergrundfarbe, um den Inhalt zu löschen. Das Löschen eines

Bildes oder eines Bildteils ist ja tatsächlich nichts weiter als das Auffüllen mit Punkten in der Hintergrundfarbe. Die acht Werte des gelöschten Musters werden in den Feldern 'Muster%' und 'Allemuster%' auf 0 gesetzt. Dann versetzen wir das noch ausgemalte "Lösch"-Kästchen in den Normalzustand zurück. Zuletzt wird das neue Muster, einfach ein leeres Feld, noch an seinem Platz innerhalb der neun Muster-Kästchen dargestellt.

Der "Inv."-Teil funktoniert sehr ähnlich. Das Invertieren können Sie benutzen, um ein Muster in Negativdarstellung zu erhalten. Für diese Funktion werden die 0er- und 1er-Bits genau umgekehrt. Die Funktion XOR &HFFFF erledigt das für uns. Denn wenn wir alle Bits mit der Zahl 1 XOR-verknüpfen, wird aus 1 der Wert 0, weil 1 XOR 1 = 0 ergibt. Und aus 0 wird 1, denn 0 XOR 1 = 1. Die neuen Werte übernehmen wir gleich wieder in 'Muster%' und 'Allemuster%'. Dann rufen wir 'Zeigmuster:' auf, um das invertierte Muster in der Arbeitsmatrix darzustellen. Alles weitere läuft nach bekanntem Schema ab: "Inv."-Kästchen löschen und zugehöriges Muster-Kästchen zeichnen.

Wurde eines der Kästchen "Load" oder "Save" angeklickt, rufen die nächsten beiden Teile einfach die Unterprogramme für 'Musterladen:' und 'Musterspeichern:' auf und springen nach der Rückkehr zurück zum Hauptprogramm.

Vom nächsten Teil, 'Markmuster:', haben wir schon gehört: Es löscht zuerst den Rahmen um das alte Muster und zeichnet dann einen neuen Rahmen um das neue Muster. Die Nummer des neuen Musters findet das Unterprogramm in der Variablen 'Musternr'. Um beim nächsten Mal den alten Wert zu kennen, belegt das Programm die Variable 'Altmuster' mit der Nummer des jetzt aktuellen Musters. Beim nächsten Wechsel ist dieses Muster ja schon wieder das alte, dessen Umrahmung dann gelöscht werden soll.

Der Teil 'Zeigmuster:' ist uns von der Funktion her auch nicht mehr unbekannt: Er bringt ein Füllmuster Punkt für Punkt in die Arbeitsmatrix. Der Aufbau des Musters in der Arbeitsmatrix dauert ca. 3 Sekunden. Damit das Unterprogramm in dieser Zeit nicht gestört wird, deaktivieren wir kurzzeitig das Event Trap-

ping. Der alte Inhalt der Arbeitsmatrix wird vorher durch Zeichnen eines Blocks in Hintergrundfarbe gelöscht. Zum Aufbau der Matrix checken wir Bit für Bit die einzelnen Werte des Felds 'Muster%' ab und zeichnen für jedes gesetzte Bit ein Kästchen. Das war's schon. Nicht vergessen, das Event Trapping wieder zuzulassen, dann erfolgt der Rücksprung. Apropos "nicht vergessen": Wann haben Sie eigentlich das letzte Mal abgespeichert?

Bleibt nur noch eine Kleinigkeit: Was hilft Ihnen die komfortabelste Muster-Definition, wenn Ihre Muster nach dem Ausschalten des Amiga verloren gehen? Aus diesem Grund haben wir die Möglichkeit vorgesehen, einen kompletten Satz aus neun Füllmustern auf Diskette abzuspeichern und von Diskette zu laden.

Der nächste Teil des Programms ist für das Laden zuständig. Auch während 'Musterladen:' unterbrechen wir Event Trapping, da der Amiga in dieser Zeit nicht gestört werden will. Der PAINT-Befehl in der zweiten Zeile des Unterprogramms malt das "LOAD"-Kästchen aus, um die Auswahl des Anwenders optisch zu bestätigen. Dann rufen wir das Unterprogramm 'Eingabename:' auf. Hier fragt Sie der Amiga nach dem Namen der Datei, die gelesen oder geschrieben werden soll. Im String 'Nam$' liefert das Unterprogramm den gefragten Namen. Falls 'Nam$' ein leerer String ist, hat der Anwender nur die <RETURN>-Taste gedrückt, ohne einen Namen einzugeben. In diesem Fall beenden wir das 'Musterladen:'-Unterprogamm und kehren zur Musterdefinition zurück. Der Teil 'EndeMLaden' wird das erledigen.

Gibt es jedoch einen Namen, wird die entsprechende Datei auf Diskette zum Lesen geöffnet. Näheres dazu und zum Befehl OPEN erfahren Sie erst im zweiten Teil dieses Buchs. Soviel verstehen Sie aber sicher jetzt schon: Die Zahlen, die die Muster festgelegen, werden Wert für Wert von der Diskette in das Feld 'Allemuster%' eingelesen. Das geschieht in den beiden ineinander verschachtelten FOR...NEXT-Schleifen. Der CLOSE-Befehl ist

eine notwendige Folge von OPEN, auch über ihn werden Sie schon bald mehr erfahren. Das war schon der Teil, der für's Laden zuständig ist. 'EndeMLaden:' beendet das Laden.

Window 5 wurde von 'Eingabename:' aufgemacht und muß nun wieder geschlossen werden. Window 4 soll wieder als Ausgabe-Window verwendet werden. Das LOAD-Feld wird in der Hintergrundfarbe ausgemalt, die farbige Füllung verschwindet also wieder. Event Trapping für Maus und Pulldowns wird wieder zugelassen. Danach bringen wir die neuen Muster auf den Bildschirm. Bisher stehen sie ja nur im Feld 'Allemuster%', der Anwender sieht noch nichts von ihnen. Dazu kopieren zwei verschachtelte Schleifen der Reihe nach die Werte für Muster 1 bis Muster 9 aus 'Allemuster%' ins Feld 'Muster%' und zeichnen damit immer eines der neun Kästchen neben der Arbeitsmatrix. Nach den beiden Schleifen wird das Feld 'Muster%' mit den Daten des Musters geladen, das der Nachfolger des zuletzt verwendeten Musters ist.

Die beiden Unterprogramme zum Speichern von Mustern sind auch nicht schwerer. 'Musterspeichern:' schaltet wieder Event Trapping aus, malt das "Save"-Feld aus, läßt das Unterprogramm 'Eingabename:' einen Namen feststellen, eröffnet die in 'Nam$' angegebene Datei zum Schreiben und schreibt Wert für Wert die Inhalte des Felds 'Allemuster%' auf Diskette. Datei wieder schließen, das war schon alles. Die Funktion von "EndeMSpeichern:' können Sie sich sicher denken: Es schließt das Speichern ab. Wie auch beim Laden muß Window 5 geschlossen und Window 4 aktiviert werden. Das "Save"-Feld wird in seinen ungefüllten Normalzustand zurückversetzt und Event Trapping wieder zugelassen, dann kommt schon der Rücksprung. Auf die Befehle CVI, MKI$ und INPUT$ gehen wir im zweiten Teil des Buchs ("Der Daten-Amiga") ausführlich ein.

Stellt sich jetzt die Frage, wie Sie das Laden und Speichern von Mustern bedienen? Sie speichern alle neun Muster gleichzeitig in einer Datei auf Diskette ab. Eine Datei ist eine Aufzeichnung wie Programme auch, nur daß sie Daten enthält. Wenn Sie das "Save"-Feld angeklickt haben, erscheint in der Mitte des Bildschirms ein Window. Es nimmt die ganze Bildschirmbreite ein,

ist aber nur eine Zeile hoch. Tippen Sie hier den Namen ein, den Sie Ihrer Muster-Datei geben wollen. Das funktioniert genauso wie beim Abspeichern von Programmen. Wählen Sie kurze, eindeutige Namen, z.B. "Muster1". Dann erkennen Sie später die Muster-Daten leichter. Mit <RETURN> bestätigen Sie Ihre Eingabe. Sie werden nun beobachten können, daß das rote Lämpchen am Diskettenlaufwerk kurz aufleuchtet. Ihre Datei wird gespeichert. Denken Sie bitte unbedingt daran, daß Sie die Diskette nicht entnehmen dürfen, während das rote Lämpchen leuchtet. Nach dem Speichern kehrt das Programm wieder zur Muster-Definition zurück, wo Sie normal weitermachen können.

Zum Laden klicken Sie das "Load"-Feld an. Das Window für die Namenseingabe erscheint wieder. Geben Sie den Namen einer Muster-Datei ein, die Sie schon auf Diskette abgespeichert haben (z.B. "Muster1"), und drücken Sie <RETURN>. Das Programm lädt nun die Muster von Diskette und zeigt sie Ihnen in den neun Muster-Feldern. Sie können diese Muster genauso verwenden wie die Standard-Muster vom Anfang. Mit der Zeit werden Sie eine größere Auswahl an Füllmustern definiert und abgespeichert haben. Daraus können Sie dann beim Zeichnen frei auswählen.

Machen wir ein bißchen im Programm weiter. Da gibt es den Programmteil 'Musteraus:'. Er hat beim Muster-Programm die gleiche Funktion wie 'Farbaus:' beim Farb-Programm: die Muster-Definition wird beendet, das Programm kehrt in den Zeichenmodus zurück. Sie erreichen das durch einen Klick ins OK-Feld. 'Musteraus:' läßt das zweite Pulldown wieder zu, das die Zeichenarten beinhaltet, und setzt den Modus auf Nummer 1 (Zeichnen). Es schließt das Window des Muster-Unterprogramms, macht Window 2, wo sich Ihr Bild befindet, zum Ausgabe-Window, aktiviert das zuletzt gewählte Muster und kehrt zum Hauptprogramm zurück.

Jetzt fehlt uns noch der Programmteil 'Eingabename:'. Er fragt beim Laden und Speichern nach einem Dateinamen. In der String-Variablen 'Altname$' merkt sich das Unterprogramm den letzten Inhalt von 'Nam$'. Dann erzeugt es das Window, in dem Sie den Namen eingeben können, und bringt mit CLS den Text-

cursor in die erste Zeile. Mit LINE INPUT fragen wir eine Eingabe ab. Wenn Sie ein =-Zeichen oder ein Sternchen eingeben, wird der zuletzt benutzte Name übernommen. Das erspart Ihnen unter Umständen viel Tipparbeit, wenn Sie modifizierte Muster unter gleichem Namen wieder abspeichern wollen. Den alten Namen erfährt das Programm aus dem String 'Altname'.

Das war's auch schon - das Programm kann dorthin zurückspringen, von wo aus es aufgerufen wurde.

Sie dürfen jetzt aufatmen. Der Marathon aus Eintippen und Verdauen von Erklärungen, den dieses Kapitel sicher für Sie darstellte, ist kurz vor seinem Ziel. Geben Sie noch den letzten Teil ein, dann ist das Malprogram komplett.

```
Abfrage:
  MENU 1,0,0 : MENU 2,0,0
  MENU OFF : MOUSE OFF
  WINDOW 5,"ACHTUNG!",(43,70)-(270,120),0,1
  COLOR 0,1 : CLS : LOCATE 2,3
  PRINT "Wollen Sie Ihr Bild"
  PRINT "  wirklich löschen?"
  PATTERN ,Voll%
  LOCATE 5,12 : PRINT "Ja";
  LOCATE 5,20 : PRINT "Nein";
  LINE (77,31)-(127,46),0,b
  LINE (145,31)-(195,46),0,b
  SOUND 880,6,100
Warte:
  Test=MOUSE(0)
  WHILE MOUSE(0)=0
    x=MOUSE(1) : y=MOUSE(2)
  WEND
  IF (y<46 AND y>31) THEN
    IF (x<127 AND x>77) THEN PAINT (79,33),3,0 : OK=1 : GOTO Endabfrage
    IF (x<195 AND x>145) THEN PAINT (147,33),3,0 : OK=0 : GOTO Endabfrage
  END IF
  GOTO Warte
```

```
Endabfrage:
  MENU ON : MOUSE ON : MENU 1,0,1 : MENU 2,0,1
  WINDOW CLOSE 5 : WINDOW 2
RETURN

Ende:
  MENU RESET
  SCREEN CLOSE 1
END
```

Im jetzt eingegebenen Teil des Programms wird die Sicherheitsabfrage durchgeführt, in der das Löschen eines Bildes oder das Beenden des Programms bestätigt werden muß. Wir bauen dazu ein Window auf, das über dem Bild erscheinen wird. Wie üblich wird vorher Event Trapping deaktiviert. Auch die Pulldowns sperren wir, damit der Anwender gleich erkennt, daß er zur Zeit nichts auswählen kann. Das Window trägt den Titel "ACHTUNG!", denn der Benutzer soll deutlich merken, daß dies seine letzte Chance ist, sein Bild zu retten. Im Window erscheint der Text "Wollen Sie Ihr Bild wirklich löschen?", und zwei Felder für "Ja" und "Nein" werden gezeichnet. Während das Window aufgebaut wird, hören Sie auch noch einen Warnton. Den erzeugt ein SOUND-Befehl, näheres über ihn erfahren Sie im dritten Teil des Buchs. Die 'Warte:'-Schleife läuft so lange, bis Sie in eines der beiden Felder geklickt haben. Je nach Ihrer Entscheidung ("Ja, löschen" oder "Nein, nicht löschen") bekommt die Variable OK den Wert 1 oder 0. Das aufrufende Programm ('Projekt:') kann dann entsprechend reagieren.

'Endabfrage:' tut das, was der Name schon ahnen läßt: Es beendet die Abfrage. In gewohnter Weise wird Event Trapping wieder aktiviert, und beide Pulldowns stehen wieder zur Auswahl.

Bleibt noch ein letzter kleiner Programmteil: 'Ende:'. Hierher springt das Programm, wenn es beendet werden soll. MENU RESET stellt die Grundbelegung der Pulldowns her, SCREEN CLOSE 1 schließt den Screen, auf dem unser Malprogamm arbeitet. END schließlich beendet das Programm. Der Befehl END hat an dieser Stelle allerdings eher kosmetische

Gründe. Kommt das Programm in die letzte Zeile und findet keinen Sprungbefehl, ist sowieso Schluß. Es gibt aber zwei Gründe, END doch zu verwenden: Erstens, um (wie in unserem Fall) das Programmende im Listing deutlich zu markieren. Und zweitens um das Programm zu beenden, obwohl im Listing noch weitere Zeilen (z.B. Unterprogramme) folgen.

Andächtige Stille. Sie sind wieder im Direktmodus, das Malprogramm ist zu Ende. Es ist mal wieder angebracht, Sie zu beglückwünschen: Auch dieses bisher längste Programm im Buch haben Sie gut hinter sich gebracht. Die Mühe hat sich aber sicher gelohnt: Mit unserem Malprogramm steht Ihnen ein leistungsfähiges Utility zur Verfügung. Im nächsten Teil werden Sie unter anderem lernen, wie Sie die Bilder, die Sie gemalt haben, auf Diskette abspeichern und später wieder einlesen können. Außerdem werden wir Ihnen Möglichkeiten zeigen, die Bilder auch in anderen Programmen zu verwenden, etwa im Videotitel-Programm.

Probieren Sie das Malprogramm jetzt nach Lust und Laune aus. Was Sie zur Bedienung wissen müssen, haben wir Ihnen im Verlauf der Erklärungen schon gesagt. Ein paar Tips werden wir Ihnen gleich noch nachliefern.

Zuerst aber noch etwas anderes: Wenn Sie das Programm selbst eingetippt haben, werden Sie beim Ausprobieren sicher Fehler feststellen. In den meisten Fällen wird das Programm abbrechen und der Fehler im LIST-Window angezeigt - dann sollten Sie den hervorgehobenen Befehl genau mit unserer Vorlage vergleichen und korrigieren. Schwieriger wird's, wenn sich das Programm an irgendeiner Stelle anders verhält, als von uns beschrieben. In diesem Fall müssen Sie wahrscheinlich ganze Programmteile überprüfen, bis Sie den Fehler gefunden haben. Ärgern Sie sich bitte nicht über sich selbst: Es ist beinahe unmöglich, 9 Seiten Listing einzugeben, ohne dabei einen Fehler zu machen. In der Endphase des Programms mußten auch wir mehr als oft versteckte Fehler suchen und ausmerzen, bis alles zufriedenstellend lief. Programmierers Leid, Programmierers

Freud. Fest steht, wenn Sie alles, was hier beschrieben wurde, verstanden haben, dann haben Sie eine ziemlich perfekte Grundlage für die Grafikprogrammierung auf dem Amiga.

Zu guter Letzt wollen wir noch ein paar Tips und Anmerkungen loswerden: Nach dem Starten dauert es einen Augenblick, bis das Programm benutzt werden kann. In dieser Zeit werden die Vorbereitungen durchgeführt. Aber schon nach wenigen Sekunden ist das Programm einsatzbereit.

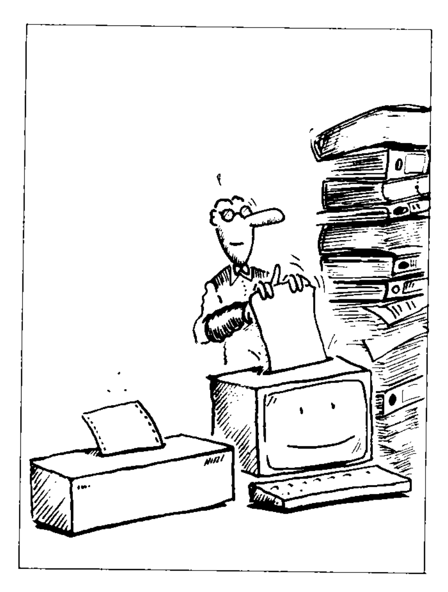
Ein Wort zum Mauszeiger: Sie werden merken, daß die Standardform des Zeigers zum Zeichnen nicht so gut geeignet ist. Das liegt hauptsächlich daran, daß Sie nicht genau sehen können, wo der Punkt ist, mit dem Sie zeichnen. Bei Bedarf können Sie sich mit Preferences einen Zeichen-Cursor nach eigenen Vorstellungen konstruieren. Ein Fadenkreuz wäre z.B. recht gut geeignet.

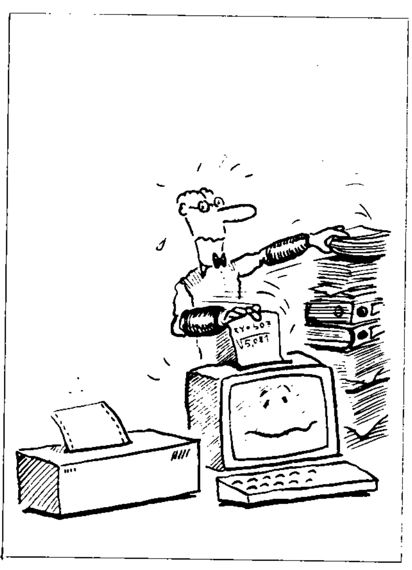
Zum Thema Muster: Sie können mit den Füllmustern Flächen füllen und dicke Linien zeichnen. Aber Achtung: Wenn das Füllmuster Löcher aufweist, können Sie auch keine umschließende Linie damit erzeugen. Falls Sie eine solche Linie als Begrenzung beim Ausmalen verwenden, läuft die Farbe aus und zerstört wahrscheinlich Teile Ihres Bildes. Deshalb sollte (wie in unserer Grundeinstellung) eines der neun Muster immer ein volles, einfarbiges Muster sein, auf das Sie bei Bedarf zurückgreifen können.

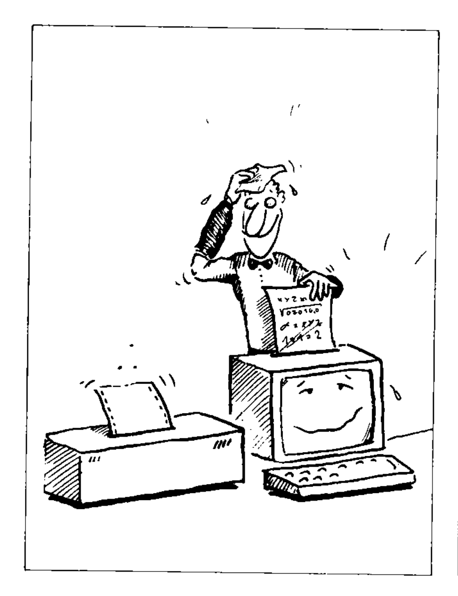
Schließlich noch etwas zum Schreiben von Texten im Bild. Diese Option sollten Sie ohnehin sparsam anwenden, denn zuviel Text in Bildern schlägt schnell die grafischen Anteile tot. Beim Textschreiben wird die Zeichenfarbe für den Text und die Ausmalfarbe für den unterlegenden Balken verwendet. So können Sie Text in gefüllte Flächen einpassen. Füllmuster kann die Text-Option jedoch nicht verwenden. Text hat beim Schreiben Priorität über die Grafik. Die Teile der Grafik, die unter dem Text liegen, werden verdeckt. Wenn Sie auf den leeren Bildschirm schreiben wollen, wählen Sie einfach vorher die Hintergrundfarbe als Ausmalfarbe.

Diese Tips sollten Ihnen helfen, das Malprogramm optimal einzusetzen. Damit ist dieses Kapitel und überhaupt der Grafik-Teil des Buchs vorbei. Bleibt uns nur noch, Ihnen viel Spaß beim Malen zu wünschen und natürlich bei all den vielen Experimenten, die Sie jetzt damit anstellen können. Natürlich nur unter der Voraussetzung, Sie haben das Programm nach dem eigenen Eintippen auch abgespei... Ach, lassen wir das - wir unterhalten uns ja jetzt schon fast von Profi zu Profi.

II. Der Daten-Amiga









3. Ein Herz für Daten - Disketten und Dateiverwaltung

Und schon sind wir im zweiten großen Teil dieses Buchs. Hoffentlich haben Sie sich vorher auch eine kleine Pause gegönnt. Die hatten Sie wirklich verdient. Nachdem wir uns bisher ausführlich mit den Themen Grafik und Animation beschäftigt haben, geht es jetzt um die Dinger, die für die Datenverarbeitung so wichtig sind: Die Daten. Ihr bevorzugter Aufenthaltsort sind Disketten. Aber auch im Speicher des Amiga trifft man eine Vielzahl von Vertretern dieser Spezies. Was man mit ihnen anfangen kann, erfahren Sie in den nächsten Kapiteln. Und nicht zuletzt springen dabei für Sie wieder einige nützliche Programme heraus.

Zunächst wollen wir uns eine geeignete Umgebung für die kommenden Experimente schaffen: Es wird höchste Zeit, AmigaBASIC und seinen Programmen eine eigene Diskette zu gönnen. Schließlich sind wir immer noch zu Gast auf der Extras-Diskette. Dort kann man uns aber nicht mehr allzu lange Gastfreundschaft gewähren.

3.1 AmigaBASIC zieht um - Anlegen einer eigenen BASIC-Diskette

Wir haben uns eigentlich recht unverschämt breit gemacht auf der Extras-Diskette. Einfach eine eigene Programm-Schublade angelegt und dann kräftig und ohne viel darüber nachzudenken unsere Programme dort hineingeschaufelt. Wenn Sie bereits viele eigene Programme abgespeichert haben, ist es aber mittlerweile ziemlich eng geworden in unserer Gast-Unterkunft. "Extras" war vorher schon recht voll: AmigaBASIC allein nimmt einigen Platz in Anspruch, die Beispiele aus der "BasicDemos"-Schublade tun ein Übriges, und die Schubladen "FD1.2" und "Tools" belegen auch ein gutes Stück Disketten-Platz.

Wir hoffen es ja nicht, aber vielleicht ist Ihnen das folgende sogar schon passiert: Wenn Sie sehr häufig verschiedene Programme abspeichern, ist die Extras-Diskette irgendwann voll. In diesem Fall erscheint zuerst ein Dialog-Window. So heißen die Windows, in denen Sie aufgefordert werden, eine bestimmte Diskette einzulegen oder etwas ähnliches zu tun. Sie werden von der Workbench erzeugt. In dem Dialog-Window steht zu lesen "Volume Extras 1.2 is full". Die Diskette "Extras 1.2" ist also dem Platzen nahe. Das Abspeichern, das diese Nachricht zur Folge hatte, kann nicht mehr korrekt durchgeführt werden. Bestätigen Sie, daß Sie den Inhalt des Windows gelesen haben, indem Sie ins Feld "Cancel" klicken. Möglicherweise läßt sich das Window beim ersten Mal davon noch nicht beirren, es taucht vielmehr ein zweites Mal auf. Das passiert manchmal und ist wahrscheinlich ein Fehler in der gegenwärtigen Version der Workbench. Wählen Sie einfach nochmal "Cancel".

Als ob es noch nicht genug wäre, bringt im Anschluß daran AmigaBASIC noch eine eigene Fehlermeldung: "Disk full". Klicken Sie wie gewohnt ins OK-Feld der Meldung.

Sollten Sie all das bereits erlebt haben, interessiert Sie sicher, was Sie tun können, um wieder ausreichend Platz zum Abspeichern Ihrer Programme zu bekommen. Und selbst wenn das Problem für Sie noch nicht aktuell ist, allzulange würde es wahrscheinlich nicht mehr dauern. Die Lösung ist ganz einfach: Wir legen eine eigene Diskette für AmigaBASIC und seine Programme an. Dazu brauchen Sie eine Leerdiskette bzw. eine Diskette, deren Inhalt gelöscht werden darf.

Die Arbeiten, die zum Anlegen einer Diskette notwendig sind, können nicht von AmigaBASIC aus erledigt werden, wir müssen auf die Workbench zurückkehren. Wenn Sie kein Programm mehr im Speicher haben, wählen Sie bitte "Quit" aus dem Project-Menü oder geben Sie den Befehl SYSTEM im BASIC-Window ein. Was aber, wenn mitten im Abspeichern eines wichtigen Programms ein "Disk full"-Error aufgetreten ist? Noch ist Ihr Programm ja nicht auf Diskette abgespeichert. Falls Sie jetzt AmigaBASIC verlassen, verlieren Sie das Programm. Also halten wir das aktuelle BASIC-Programm einfach im Speicher. Dank

der Multitasking-Fähigkeit des Amiga ist das problemlos möglich. Klicken Sie das LIST-Window aus und verkleinern Sie das BASIC-Window auf die kleinstmögliche Größe. Verschieben Sie es dann an eine Stelle des Bildschirms, wo es Sie nicht weiter stört. Jetzt läuft AmigaBASIC im Hintergrund, und Sie können mit der Workbench arbeiten.

Um die Workbench zu aktivieren, klicken Sie bitte an eine beliebige Stelle außerhalb des BASIC-Windows. Also sozusagen einfach ins Blaue. In der Kopfleiste erscheint der Text "Workbench release 1.2:" und dahinter der Speicherplatz, der Ihnen zur Verfügung steht.

Entnehmen Sie bitte "Extras" aus dem eingebauten Laufwerk und legen Sie Ihre Leerdiskette ein. Zuerst müssen wir die neue Diskette formatieren. Wenn Sie das jetzt zum ersten Mal machen, wissen Sie wahrscheinlich gar nicht, wozu es überhaupt nötig ist. Deshalb eine kurze Erklärung.

Eine Diskette ist im Prinzip nichts weiter als eine Plastikscheibe in einer Umhüllung. Ihr Amiga benötigt sogenannte 3½ Zoll-Disketten. Das bedeutet, daß die Disketten einen Durchmesser von 3½ Zoll (8,9 cm) haben. Solche Disketten werden in fast allen Computern verwendet, die in letzter Zeit auf den Markt kamen. Noch vor wenigen Jahren wurden für Home- und Personal-Computer vorwiegend 5¼ Zoll-Disketten (Durchmesser 13,3 cm) benutzt. Die sind aber empfindlicher und nicht so einfach zu transportieren. Die Plastikscheibe im Inneren der Diskette kann magnetisiert werden. Das erledigt ein Magnetkopf, der Schreib-/Lesekopf heißt, weil er Bits auf die Diskette schreibt und sie dann später wieder liest. Positive bzw. negative Magnetisierung entsprechen "Bit an" bzw. "Bit aus". Aus den Bits setzen sich Bytes zusammen und aus den Bytes Daten oder Programme.

Die Bits werden auf konzentrischen (= ineinanderliegenden) Spuren aufgezeichnet. Eine Diskette ist also vom technischen Hintergrund her eine Kombination aus Schallplatte und Magnetband-Kassette. Die Spuren müssen vor dem ersten Gebrauch der

Diskette vorbereitet werden, damit sich die Schreib-/Leseköpfe des Diskettenlaufwerks später zurechtfinden. Ein "Format" wird erzeugt, und dazu dient das Formatieren.

Vielleicht fragen Sie sich, warum die Spuren nicht von vornherein auf der Diskette sind. Das liegt daran, daß Sie 3½ Zoll-Disketten für verschiedene Computer verwenden können. Verschiedene Computer haben aber verschiedene Disketten-Formate. Obwohl die Disketten dieselben sind, werden die Informationen in unterschiedlicher Weise aufgezeichnet. Die Anzahl der Spuren, die Abstände der Spuren, die Codierung der Informationen und die Steuer-Informationen auf der Diskette unterscheiden sich so sehr voneinander, daß z.B. ein Atari ST-Computer mit einer Amiga-Diskette nicht das geringste anfangen kann. Er kann das Format der Diskette nicht verarbeiten. Deshalb muß jeder Computer seine Disketten so vorbereiten, daß er damit arbeiten kann.

Ihr Amiga kann verschiedene Diskettenformate verarbeiten. Im Normalfall werden die Disketten unter *AmigaDOS* verwaltet. Das Kürzel DOS spricht sich wie Boß, nur mit D am Anfang. Es steht für den englischen Ausdruck "Disk Operating System" (Disketten-Betriebssystem). AmigaDOS ist ein Programm, das für die Steuerung und Verwaltung der Ein- und Ausgabe bei Disketten zuständig ist. Wenn auf dem Amiga ein anderes DOS läuft, kann er auch andere Diskettenformate lesen. Der Amiga kann zum Beispiel durch eine softwaremäßige MS-DOS-Anpassung das Diskettenformat des IBM PC verarbeiten. Solange Sie sich nur mit AmigaBASIC beschäftigen, brauchen Sie die fremden Formate aber nicht zu interessieren.

Für diejenigen unter Ihnen, die gern technische Daten wissen möchten: Das AmigaDOS-Diskettenformat verwendet 80 Spuren, benutzt dazu beide Diskettenseiten und kann 880 KByte pro Diskette speichern. Beide Seiten der Diskette werden beschrieben, dazu haben die Diskettenlaufwerke des Amiga zwei gegenüberliegende Schreib-/Leseköpfe. Deshalb sollten Sie beim Kauf übrigens darauf achten, daß Sie "Double Sided"-Disketten kaufen, also Disketten, die auf beiden Seiten geprüft sind. Wenn Sie eine unformatierte Diskette eingelegt haben, ist auf der Work-

bench-Oberfläche ein Disketten-Icon erschienen, das den Namen "DF0:BAD" hat, bzw. "DF1:BAD" oder "DF2:BAD", wenn Sie ein externes Diskettenlaufwerk besitzen und die Leerdiskette dort eingelegt haben. Was, bitteschön, hat das "BAD" jetzt wieder zu bedeuten? Da AmigaDOS die Diskette nicht lesen kann, bezeichnet es sie als "BAD" (schlecht, fehlerhaft). Solche Disketten sind entweder überhaupt nicht formatiert oder sie haben ein Format, das AmigaDOS nicht kennt. Auch MS-DOS-Disketten bezeichnet AmigaDOS als "BAD".

Achten Sie bitte unbedingt darauf, daß die Diskette, die formatiert werden soll, keine wichtigen Programme oder Daten enthält. Der alte Inhalt geht unwiederbringlich verloren. Damit Sie immer wissen, welchen Inhalt eine Diskette hat, sollten Sie sie mit einem Aufkleber kennzeichnen, auf dem mindestens der Name der Diskette steht. Solche Aufkleber werden bei Leerdisketten mitgeliefert.

Wenn Sie nach all unseren Erklärungen und Warnungen sicher sind, daß Sie die gerade im internen Laufwerk liegende Diskette formatieren wollen, dann aktivieren Sie jetzt bitte das Icon der neuen Diskette. Klicken Sie einmal auf das Symbol, so daß die Diskette schwarz wird. Wählen Sie dann aus dem Disk-Pulldown den Menüpunkt "Initialize". Die Begriffe "initialisieren" (übersetzt etwa: vorbereiten) und "formatieren" werden beide für den gleichen Vorgang verwendet, nämlich das Schreiben der Spuren auf eine neue Diskette. Ein Window erscheint, in dem Sie aufgefordert werden, die Workbench-Diskette einzulegen. Wenn Sie zwei Diskettenlaufwerke besitzen, sollte sich die Workbench im zweiten Laufwerk befinden. Bei nur einem Laufwerk entnehmen Sie bitte die Leerdiskette und legen Sie die Workbench ein.

Nach kurzem Laden bittet Sie der Amiga, die Leerdiskette zurück ins Laufwerk zu legen. Zur Sicherheit fragt er aber nach: "OK to Initialize disk in drive 0 (all data will be erased) ?" oder auch "OK to Initialize disk ..." gefolgt vom Namen der Diskette. Sie sollen also bestätigen, daß Sie die Diskette im Laufwerk 0 formatieren wollen. Und Sie sind einverstanden, daß dabei alle Daten auf der Diskette gelöscht werden. Zu den Laufwerksbezeichnungen: Das interne Laufwerk bezeichnet AmigaDOS als

Laufwerk (engl.: drive) 0, das erste Zusatz-Laufwerk hat die Nummer 1. Wenn Sie einen Amiga 2000 besitzen und nur ein Diskettenlaufwerk eingebaut haben, bezeichnet AmigaDOS Ihr externes Laufwerk allerdings trotzdem als DF2:. Wenn Sie damit nicht glücklich werden können, empfehlen wir Ihnen, unser "Das große Amiga-2000-Buch" zu lesen. Dort finden Sie unter anderem auch einen Trick, wie man die Problematik mit den Laufwerksbezeichnungen umgehen kann. Nun aber genug der Werbung und weiter mit dem Formatieren unserer Leerdiskette.

Wenn Sie ins Feld "Continue" (Weitermachen) klicken, beginnt unwiderruflich die Formatierung. Die 80 Spuren werden auf die Diskette geschrieben und dabei gleich geprüft. Im "Initialize"-Window steht, welche Spur gerade bearbeitet wird und wieviele noch zu erledigen sind. Die erste Spur ist Spur Nummer 0, die letzte Spur hat die Nummer 79. Das Wort "Formatting" ist zu lesen, während die Spur angelegt wird. "Verifying" bedeutet, daß sie überprüft wird.

Sollte beim Überprüfen ein Fehler festgestellt worden sein, erscheint eine entsprechende Meldung. Sie kann verschiedene Ursachen haben. Wiederholen Sie den Formatierungsvorgang noch ein- oder zweimal. Wenn es gar nicht klappt, hat vielleicht die Diskette einen mechanischen Fehler. Probieren Sie eine andere Diskette aus. Treten auch hier Probleme auf, sollten Sie sich an Ihren Händler wenden. Wir wollen aber hoffen, daß alles richtig funktioniert. Das ist ja auch der Normalfall.

Wenn die 80 Spuren angelegt sind, ist die Formatierung aber noch nicht zu Ende. Das teilt Ihnen auch Ihr Amiga mit, indem er ins "Initialize"-Window schreibt: "WARNING: Initialize Still In Progress. DO NOT REMOVE DISK." Auf Deutsch: Die Initialisierung (Formatierung) ist noch im Gange, Diskette nicht entnehmen! Die Laufwerkslampe kann zwar für einige Sekunden ausgehen, warten Sie aber trotzdem! Kurz darauf werden nämlich noch wichtige Steuerinformationen auf die Diskette geschrieben. Dann erst ist der Amiga mit allem fertig. Die formatierte Diskette trägt jetzt den Namen "Empty". Das heißt "leer", und die Diskette ist ja in der Tat auch fast leer.

Klicken Sie die "Empty"-Diskette an. Ein einsamer Mülleimer (= Trashcan) befindet sich im entstandenen Window. Die Füllanzeige auf der linken Seite zeigt an, daß die Diskette ganz leer ist. Wir wollen sie jetzt Schritt für Schritt mit ihrem Inhalt füllen.

Sicher sind Sie mit uns einer Meinung, daß "Empty" auf Dauer kein sehr passender Name für Ihre BASIC-Diskette ist. Also ändern wir ihn. Sollte das Icon der "Empty"-Diskette nicht mehr schwarz sein, aktivieren Sie es bitte wieder. Wählen Sie dann "Rename" aus dem "Workbench"-Pulldown. Eine schmale Eingabezeile erscheint in der Mitte des Bildschirms. Klicken Sie einmal mit der Maus in diese Zeile, dann können Sie den neuen Namen eintippen. Um das Wort "Empty" zu löschen, drücken Sie bitte mehrmals die <DEL>-Taste.

Wir brauchen einen sinnvollen Diskettennamen. Wie wäre es mit "BASICDisk"? Geben Sie diesen Namen bitte ein und drücken Sie <RETURN>. "Empty" wird in "BASICDisk" umgetauft. Beachten Sie, daß sich auch der Window-Titel sofort ändert.

Schließen Sie das "BASICDisk"-Window bitte noch nicht. Wir wollen jetzt verschiedene Programme auf die neue Diskette kopieren. Zuerst kommt AmigaBASIC an die Reihe. Legen Sie also bitte "Extras" ein. Mit zwei Laufwerken sind Sie jetzt natürlich wesentlich komfortabler dran: Entnehmen Sie einfach die Workbench aus Drive 1 (dem zweiten Laufwerk) und legen Sie dort die "Extras"-Diskette ein. Besitzen Sie nur ein Laufwerk, werden Sie zum Kopieren mehrmals Ihre Disketten wechseln müssen. Vergessen Sie bitte keinesfalls: Disketten dürfen erst aus dem Laufwerk entnommen werden, wenn die rote Lampe erloschen ist. Andernfalls riskieren Sie kompletten Datenverlust.

Klicken Sie das "ExtrasD"-Icon an. Das zugehörige Window erscheint auf der Workbench. Bewegen Sie dann den Mauszeiger auf das Icon von AmigaBASIC, drücken Sie die linke Maustaste und bewegen Sie den Mauszeiger mit gedrücker Taste ins "BASICDisk"-Window. Mit der Maus bewegen Sie jetzt eine Kopie des AmigaBASIC-Icons. Lassen Sie dann die Maustaste los.

Um Objekte auf der Workbench zu kopieren, verschieben Sie einfach das Icon aus dem alten Window ("ExtrasD") in das neue Window ("BASICDisk").

Wenn Sie zwei Laufwerke besitzen, wird in einem Durchgang kopiert. Während des eigentlichen Kopiervorgangs nimmt der Mauszeiger die Form des bekannten Schlaf-Wölkchens an. Bei nur einem Laufwerk müssen Sie dreimal zwischen "Extras" und "BASICDisk" wechseln. Die Workbench zeigt Ihnen in einem Dialog-Window, welche Diskette sie jeweils benötigt. Ist das Kopieren beendet, erscheint auch im "BASICDisk"-Window das AmigaBASIC-Icon.

Jetzt kommen die BASIC-Programme an die Reihe. Sie können eine Schublade, die Programme enthält, genauso kopieren wie ein einzelnes Programm. Je nachdem, wieviel Speicher Sie zur Verfügung haben und wie voll die Schublade ist, wird das bei einem einzelnen Laufwerk einige Diskettenwechsel erfordern. Kopieren Sie bitte Ihre Programm-Schublade von der "Extras"- auf die "BASICDisk"-Diskette. Übrigens: Es wird wirklich kopiert. Das heißt, es ist nicht so, daß die Schublade mit Ihren BASIC-Programmen von einer Diskette auf die andere gebracht wird. Sie ist noch immer auf der "Extras" vorhanden. Sobald Sie irgendetwas von einem Diskettenfenster auf ein anderes legen, wird kopiert, das Programm ist also später auf beiden Disketten vorhanden. Anders ist es, wenn Sie innerhalb einer Diskette bestimmte Dinge zum Beispiel von einer Schublade in eine andere legen. Dann wird nicht kopiert, sondern richtig verschoben. Das aber nur als Hintergrundinformation für Sie.

Jetzt haben Sie erstmal Ihre eigenen Werke und die bisherigen Beispielprogramme umgelagert. Die Programme befinden sich aber noch sehr ungeordnet in ihrer Schublade. Wir sollten die Möglichkeiten ausnutzen, die uns die Workbench zum Aufräumen bietet.

Das "BASICDisk"-Window lassen Sie bitte geöffnet. Schließen Sie aber das "ExtrasD"-Window und öffnen Sie dafür das Workbench-Window. Wie Sie schon wissen, werden Besitzer eines einzelnen Laufwerks aufgefordert, die Workbench-Diskette

einzulegen. Solche Aufforderungen erwähnen wir ab jetzt nicht mehr, da es dabei keine Unklarheiten mehr geben dürfte. Vertrauen Sie einfach auf Amiga - er weiß schon, was für ihn gut ist.

Im einem vorhergehenden Zwischenspiel haben wir die Schublade für Ihre BASIC-Programme angelegt. Sie können sich vielleicht noch erinnern, wie das geht: Im Workbench-Window befindet sich eine Schublade namens "Empty". Wenn Sie neue, leere Schubladen benötigen, kopieren Sie einfach die "Empty"-Schublade. Verschieben Sie also das Icon mit gedrückter linker Maustaste vom alten Window in das neue und lassen Sie dann los. Das Kopieren von Objekten auf der Workbench haben Sie jetzt schon so oft ausgeführt, daß wir es Ihnen von nun an auch nicht mehr erklären müssen.

Danach klicken Sie das Workbench-Window bitte aus, wir brauchen es jetzt nicht mehr. Alle verbleibenden Aufräum-Arbeiten führen wir auf unserer neuen "BASICDisk" aus.

Da wir in Zukunft, soweit es AmigaBASIC betrifft, fast ausschließlich mit dieser Diskette arbeiten wollen, sollten wir schon ein wenig Mühe darauf verwenden, sie sinnvoll und übersichtlich anzulegen. Wenn Sie Ihre Wohnung einrichten, wollen Sie ja auch auf zukünftige Neuanschaffungen vorbereitet sein. Oder stellen Sie jedesmal Ihre Bücherregale um, nur weil Sie sich ein neues Buch gekauft haben? Wir haben uns Gedanken gemacht, was an Programmen so alles entstehen könnte. Folgende Schubladen sollten angelegt werden:

- Videotitel
- Grafik
- Malprogramm
- Daten
- Sprache

- Musik

- Verschiedenes

Insgesamt benötigen wir sieben Schubladen. Auf sie wollen wir den gegenwärtigen Inhalt von "Meine Programme" (oder wie Ihre bisherige Programmschublade eben hieß) und zukünftige Programme verteilen.

Das Zauberkunststückchen "Aus eins mach sieben" beherrscht Ihr Amiga ganz souverän: Aktivieren Sie die "Empty"-Schublade und wählen Sie "Duplicate" (duplizieren, verdoppeln) aus dem "Workbench"-Pulldown. Wenige Augenblicke später erscheint die neue Schublade "copy of Empty". Jetzt haben wir schon zwei. Um die Übersicht zu behalten, versehen Sie bitte gleich die einzelnen Schubladen mit ihren späteren Namen. Dazu aktivieren Sie die jeweils neue Schublade und wählen "Rename" aus dem Workbench-Pulldown. Da Sie vor der Eingabe des neuen Namens jedesmal den Text "Empty" oder "copy of Empty" löschen müssen, verraten wir Ihnen an dieser Stelle einen Trick: Anstatt den alten Text Zeichen für Zeichen mit <DEL> zu löschen, können Sie die Tastenkombination <offene Amiga-Taste><x> verwenden. Schieben Sie die gerade bearbeitete Schublade etwas zur Seite, damit wir genug Platz für die nächste haben. Nach dem gleichen Rezept erzeugen Sie jetzt bitte fünf weitere Schubladen.

Wenn Sie damit fertig sind, liegen die Schubladen wahrscheinlich kreuz und quer im "BASICDisk"-Window herum. Durch Verschieben mit dem Mauscursor können Sie die Icons nach Belieben anordnen. Wenn Sie sich an unser zweites Zwischenspiel erinnern oder Erfahrungen von vorherigen Arbeiten auf der Workbench haben, wissen Sie, daß derartige Aufräumaktionen wenig Eindruck auf den Amiga machen, solange Sie ihnen nicht den nötigen Nachdruck verleihen. Das erreichen Sie, indem Sie mit gedrückter <SHIFT>-Taste das Icon der "BASICDisk" sowie alle Icons innerhalb des Windows aktivieren und dann die Option "Snapshot" aus dem "Special"-Pulldown wählen.

Das Bild zeigt Ihnen, wie unser "BASICDisk"-Window nach diesen Arbeiten aussieht. Natürlich dürfen Sie Ihr Window auch anders einteilen.

Ihnen wird wahrscheinlich auffallen, daß diese Schubladenanordnung exakt der Einteilung der Diskette entspricht, die wir diesem Buch beigelegt haben. Das ist auch kein Wunder. Genau nach dem von uns beschriebenen Vorbild haben wir unsere Diskette im Buch nämlich organisiert. Sie sollten trotzdem für Ihre eigenen Programm eine eigene Diskette anlegen. Und wenn Sie die Namensgleichheit der beiden Disketten stört, dann nennen Sie doch Ihre Diskette einfach anders. Wie wäre es zum Beispiel mit "Meine BASICDisk"?

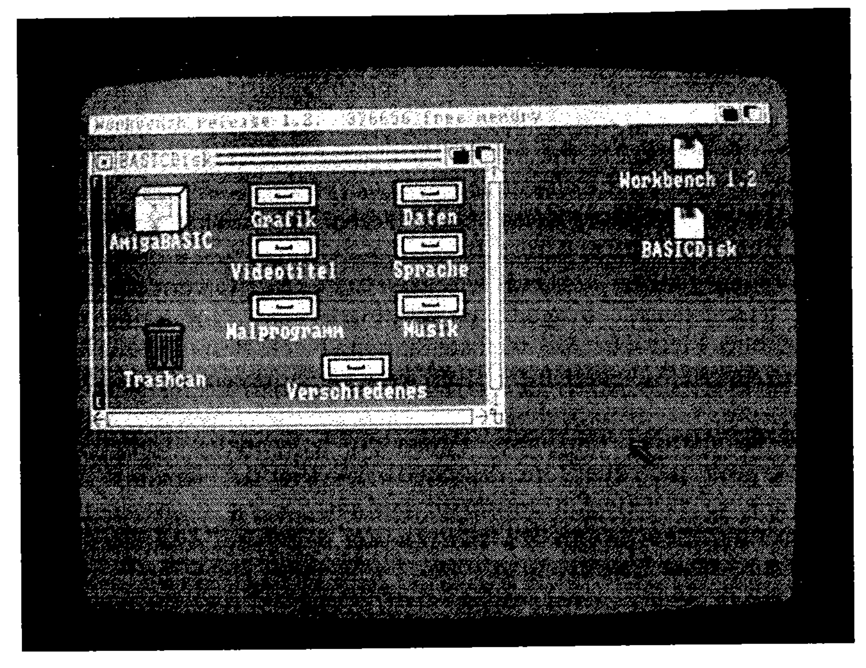


Bild 12: **Das Window der BASICDisk**

Um bei unserem Vergleich von vorhin zu bleiben: Die Möbel sind jetzt aufgebaut. Als nächstes packen wir die Dinge aus, die wir in den Schubladen unterbringen wollen.

Unsere Umzugskiste ist die Schublade "Meine Programme". Öffnen Sie diese Schublade, aber wundern Sie sich bitte nicht, wie unordentlich die einzelnen Icons darin herumliegen. Sie wissen ja: Freiwillig ist der Amiga nicht gerade ein Ordnungsfanatiker. Ihre nächste Aufgabe besteht darin, die einzelnen Programme den Schubladen zuzuordnen, wo sie thematisch am besten hineinpassen. Bringen Sie das Ballprogramm (unsere Kostprobe aus dem ersten Kapitel) zum Beispiel in die Grafik-Schublade. Die Workbench sorgt dafür, daß die Programme auf der Diskette von einer Schublade in die andere umkopiert werden. Falls Sie seit Beginn dieses Kapitels im Hintergrund noch ein BASIC-Programm stehen haben, wollen wir das jetzt auch endlich loswerden. Dazu ist ein Vorgriff auf einen BASIC-Befehl nötig, den Sie aber sowieso demnächst ausführlich kennenlernen werden. Vergrößern Sie das BASIC-Window, das sich ja immer noch auf der Workbench-Oberfläche befindet, und aktivieren Sie es. Geben Sie dann den Befehl

```
chdir "BASICDisk:
```

ein, gefolgt vom Namen der Schublade, in der das Programm abgelegt werden soll. Für ein Grafikprogramm sieht das zum Beispiel so aus:

```
chdir "BASICDisk:Grafik"
```

Danach können Sie mit dem SAVE-Befehl oder der entsprechenden Pulldown-Option Ihr Programm abspeichern. Dann klicken Sie AmigaBASIC bitte ganz aus.

Zurück auf der Workbench verteilen Sie bitte die restlichen Programme auf die sieben Schubladen. In die Schublade "Videotitel" kommt das Videotitel-Programm, alle dazugehörigen Bobs sowie die Grafikprogramme, die Sie mit dem Videotitel-Programm mergen wollen. In "Grafik" haben wir unser Balken/Torten-Utility untergebracht, außerdem alle Beispielprogramme aus dem ersten Teil des Buchs, soweit sie Grafik oder Animation betreffen. Die "Malprogramm"-Schublade beinhaltet das Malprogramm und die Muster-Dateien, die Sie beim Ausprobieren eventuell abgespeichert haben. "Daten", "Musik" und "Sprache" bleiben

zunächst noch leer (aber wir hoffen, die Spannung steigt schon wieder, was da wohl zum Schluß drin sein wird), und in der Schublade "Verschiedenes" können Sie alles unterbringen, was sonst nirgends hinpaßt. Das alles ist natürlich nur dann nötig, wenn Sie einige oder alle Programme aus unserem Buch selbst eingetippt haben. Die Programme von unserer Diskette im Buch auf Ihre Arbeitsdiskette zu kopieren ist nicht nötig und wäre auch kaum sehr sinnvoll.

Wenn Sie alle Programme verteilt haben, klicken Sie bitte die einzelnen Schubladen an und räumen Sie mit der Snapshot-Funktion in den zugehörigen Windows auf. Nachdem Sie auch das erledigt haben, ist Ihre erste eigene BASIC-Diskette angelegt. Mit ihr werden wir ab jetzt arbeiten.

Schließen Sie bitte alle Windows außer dem "BASICDisk"-Window und klicken Sie AmigaBASIC an. Im nächsten Kapitel werden Sie nämlich die Befehle zur Arbeit mit Disketten kennenlernen, die Ihnen AmigaBASIC bietet.

3.2 Von Bäumen, Affen und rosa Elefanten - BASIC-Befehle zur Arbeit mit Disketten

Um unsere Diskette zu formatieren und einzurichten, mußten wir auf die Workbench zurückkehren. AmigaBASIC unterstützt diese Funktionen nicht. Trotzdem gibt es eine stattliche Anzahl an Befehlen zur Arbeit mit Disketten. Einige haben Sie schon kennengelernt: Um Programme abzuspeichern, verwenden Sie SAVE, und um sie wieder zu laden, gibt es LOAD. Natürlich stehen Ihnen fürs Laden und Speichern auch Pulldown-Optionen zur Verfügung. Aber sie sind eigentlich auch nur getarnte BASIC-Befehle: Egal, ob Sie "Open" aus dem Project-Pulldown wählen oder LOAD im BASIC-Window eintippen - das Resultat ist in beiden Fällen das gleiche.

Die erste neue Funktion, die wir Ihnen vorstellen wollen, ist wichtig, wenn Sie die Übersicht behalten wollen. Denn nach einiger Zeit erinnern Sie sich vielleicht nicht mehr an all die Namen, die Sie den Programmen beim Abspeichern gegeben haben.

Sie können zwar auf der Workbench-Oberfläche nachschauen, aber diese Methode ist sehr umständlich und kostet Zeit und Speicherplatz. Einfacher haben Sie es mit einem BASIC-Befehl. Probieren Sie ihn gleich mal im BASIC-Window aus:

```
files
```

"Files" heißt übersetzt "Dateien, Akten". Der Befehl FILES zeigt die Namen aller Dateien im aktuellen Disketten-Inhaltsverzeichnis an. Der Begriff "Datei" beinhaltet dabei alles, was Sie auf Diskette abspeichern können, also Programme, Daten, Bilder, Schriften usw. Und was verstehen wir unter dem "aktuellen Inhaltsverzeichnis"? Schauen Sie sich die Bildschirmausgabe an, die wir erhalten haben. In der ersten Zeile lesen Sie

```
Directory of: [BASICDisk]
```

AmigaBASIC teilt Ihnen mit, welches Inhaltsverzeichnis angezeigt wird. Das englische Wort "Directory" heißt auf Deutsch "Verzeichnis, Telefonbuch". "Directory" wird in der Computerliteratur oft verwendet, wenn das Disketten-Inhaltsverzeichnis gemeint ist. Verglichen mit der Workbench, wo man erstmal einen Überblick über alles bekommt, sehen Sie zur Zeit nur das Verzeichnis, das dem Inhalt des "BASICDisk"-Windows entspricht.

Außer den Programmnamen (z.B. "AmigaBASIC") erscheinen noch andere Texte. Wenn Sie genau aufgepaßt haben, haben Sie sicher bemerkt, daß die meisten Namen paarweise auftreten. Zu "AmigaBASIC" gehört zum Beispiel eine weitere Datei namens "AmigaBASIC.info". Um die Aufgabe dieser Info-Dateien zu verstehen, müssen Sie ein wenig mehr über die Funktionsweise der Workbench wissen. Die Workbench stellt jede Datei auf der Diskette durch ein Icon dar. Das Aussehen der Icons ist nicht fest definiert. Sie haben sicher schon manche Programme gesehen, die ein eigenes, individuelles Icon besitzen. Vielleicht haben Sie auch schon mit dem Programm IconEd gearbeitet, das Sie auf der Workbench-Diskette finden. Mit IconEd können Sie das

Aussehen von Icons beliebig verändern. Es spielt überhaupt keine Rolle, für welche Datei das Icon steht, Sie können ein beliebiges Symbol dafür entwerfen.

Da zu jedem Programm ein eigenes Icon gehört, muß dessen Aussehen aber auch irgendwo abgespeichert sein. Genau dafür gibt es die Info-Dateien. Sie beinhalten diese Information. Außerdem gibt die Info-Datei Auskunft über die Lage des Icons innerhalb seines Windows und einige weitere Informationen, die von der Workbench benötigt werden.

Im Inhaltsverzeichnis tauchen außerdem Namen auf, die in eckigen Klammern stehen. Es handelt sich um Unter-Inhaltsverzeichnisse, sie entsprechen den Schubladen auf der Workbench. Auch zu diesen Namen gibt es Info-Dateien, in ihnen ist das Aussehen des Schubladen-Icons gespeichert. Das Prinzip von Unter-Inhaltsverzeichnissen ist ganz einfach: Innerhalb eines Inhaltsverzeichnisses können sich weitere Inhaltsverzeichnisse befinden. Und innerhalb eines solchen Unter-Inhaltsverzeichnisses weitere Unter-Inhaltsverzeichnisse. Das Ganze können Sie fortführen, solange es Ihnen Spaß macht und sinnvoll erscheint.

Um untergeordnete Inhaltsverzeichnisse grafisch zu erklären, sind sogenannte Baum-Strukturen sehr beliebt. Sollten Sie jetzt etwas ungläubig auf das Bild schauen, müssen wir zugeben, daß es eher an einen Baum erinnert, der einen gekonnten Kopfstand vorführt. Trotzdem zeigt so ein Schema recht gut, wie man sich aus einem übergeordneten Verzeichnis durch Unterverzeichnisse immer weiter zum gesuchten Punkt durchwählen kann.

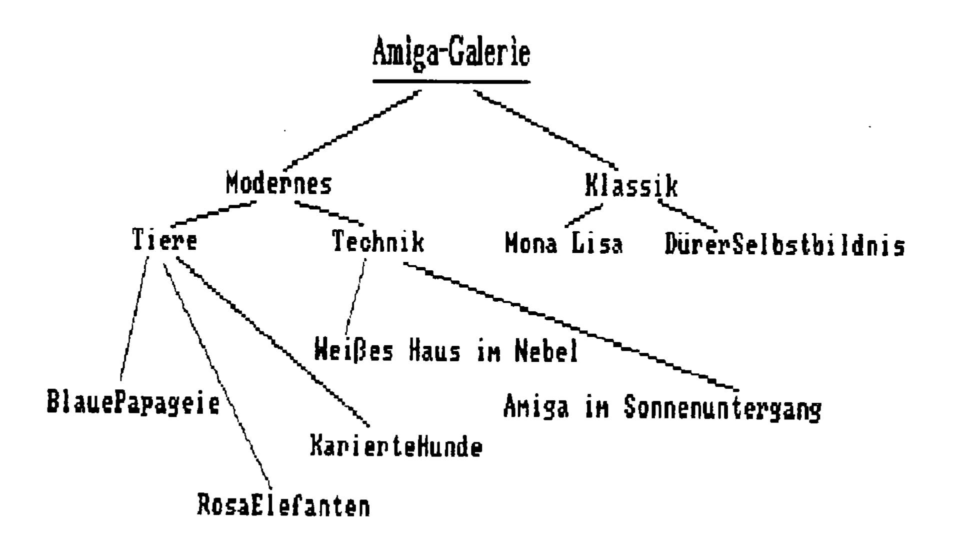


Bild 13: Die Baumstruktur eines Disketteninhaltsverzeichnisses

Nehmen wir an, wir haben eine Diskette namens "AmigaGalerie", auf der Bilder abgespeichert sind. Der Disketten-Name ist sozusagen der Baumstamm. Auf der Diskette gibt es die beiden Inhaltsverzeichnisse "Klassik" und "Modernes" - der Baum verzweigt sich in zwei große Äste. Wollen Sie z.B. das Bild "RosaElefanten" laden, führt Ihr Weg über das Unter-Verzeichnis (den Ast) "Modernes" weiter über das Unter-Unter-Verzeichnis (den Zweig) "Tiere" zum Ziel.

Sie können immer nur Programme oder Dateien aus dem aktuellen Inhaltsverzeichnis laden. Liegt das Programm an einer anderen Stelle der Diskette, wird es von AmigaBASIC nicht gefunden. Die Kennzeichnung durch Inhaltsverzeichnisse ist so eindeutig, daß Sie in verschiedenen Verzeichnissen für unterschiedliche Programme sogar denselben Namen verwenden dürfen. Deshalb ist es wichtig, das aktuelle Inhaltsverzeichnis jederzeit von AmigaBASIC aus verändern zu können.

Um diese Theorie in die Praxis umsetzen zu können, brauchen wir einen BASIC-Befehl, den Sie im letzten Kapitel schon einmal kurz gesehen haben: CHDIR heißt "Change Directory" - "Wechsle das Inhaltsverzeichnis". Wenn Sie z.B. die Namen aller Programme sehen wollen, die sich auf Ihrer "BASICDisk" im Unter-Inhaltsverzeichnis "Grafik" befinden, geben Sie ein:

```
chdir "Grafik"
```

Damit haben Sie vom übergeordneten Verzeichnis "BASICDisk" ins Verzeichnis "Grafik" gewechselt. Wenn Sie sich dieses Inhaltsverzeichnis durch

```
files
```

zeigen lassen, sehen Sie in der ersten Zeile den neuen Directory-Namen.

```
Directory of: [Grafik]
```

Darunter folgen die Namen aller Programme, die in der Grafik-Schublade liegen. Wenn wieder ein Inhaltsverzeichnis dabei wäre, könnten Sie mit CHDIR noch weiter herabsteigen (oder um beim Bild des Baums zu bleiben: hinaufklettern).

Aber was können Sie tun, wenn Sie feststellen, daß sich das gesuchte Programm nicht im aktuellen Verzeichnis befindet? Wenn Sie also sozusagen auf dem falschen Ast sitzen? Prinzipiell bieten sich Ihnen zwei Möglichkeiten: Entweder Sie gehen genau auf dem Weg zurück, auf dem Sie gekommen sind, oder Sie springen vom Baum runter und erklettern ihn von neuem. In die Welt von AmigaBASIC übersetzt ergibt das folgende Befehlsvarianten: Mit

```
chdir "/"
```

kehren Sie auf die nächsthöhere Verzeichnisebene zurück. In unserer "AmigaGalerie" kommen Sie damit z.B. vom Verzeichnis "Tiere" zurück ins Verzeichnis "Modernes". Bei erneuter Eingabe von CHDIR "/" würden Sie im Basis-Inhaltsverzeichnis "Amiga-Galerie" laden. So können Sie in neue Unter-Verzeichnisse ge-

langen und danach wieder auf die höheren Ebenen zurückkehren. Falls Sie allerdings mit CHDIR "/" aus der höchsten Verzeichnis-Ebene noch weiter zurückkehren wollen, wird sich der Amiga mit einem "File not found"-Error dagegen wehren. Es kann auch vorkommen, daß der Weg über die einzelnen Verzeichnisse zu langwierig ist oder Sie sich ganz einfach verirrt haben. Benutzen Sie in diesem Fall folgende Variante:

```
chdir ":"
```

So wählen Sie das Basis-Inhaltsverzeichnis aus, also das erste, allen anderen übergeordnete Inhaltsverzeichnis der Diskette. Von dort aus können Sie sich dann wieder in die Unter-Verzeichnisse wählen. Fehlt zum Thema Disketten-Inhaltsverzeichnisse nur noch eines: Bisher haben wir immer nur mit einer Diskette gearbeitet, nämlich unserer "BASICDisk". Was aber, wenn Sie ein zweites Disketten-Laufwerk haben? Für diesen Fall gibt es die Möglichkeit, das gewünschte Laufwerk auszuwählen. Wollen Sie sich z.B. das Basis-Inhaltsverzeichnis von Laufwerk 1 ansehen (Sie wissen ja noch: 0 ist das interne und 1 das externe), geben Sie ein:

```
chdir "df1:"
```

AmigaBASIC nennt das interne Laufwerk "DF0:" und das erste externe Laufwerk "DF1:". Wenn Sie sogar drei Laufwerke besitzen, trägt das dritte die Bezeichnung "DF2:". Es sei denn, Sie haben einen Amiga 2000 und nur ein internes Laufwerk. Dann heißt das externe nämlich DF2:.

Das D steht dabei für "Drive" (Deutsch: Laufwerk) und F für "Floppy" (Ihre Disketten heißen ja auch "Floppy Disks"). Nebenbei: Wenn Sie irgendwann einmal ein Festplattenlaufwerk besitzen, wird es "DH0:" heißen (Drive Harddisk 0). Es sei denn, Sie haben einen Amiga 2000 und steuern die Festplatte über eine eingebaute PC-Karte an. Dann heißt die Festplatte nämlich JH0:. Wenn Sie jetzt das Gefühl haben, daß der Amiga 2000 immer aus der Reihe tanzen muß, dann können wir dem nicht völlig widersprechen. Um ihn andererseits ein wenig zu verteidigen, muß man sagen, daß ein so komplexer und vielseitiger Computer

wie gerade der Amiga 2000 eben auch etwas komplexere Möglichkeiten und Bezeichnungen bedingt. Und wir werden jetzt an dieser Stelle auch nicht darauf hinweisen, daß all diese Dinge in unserem "Das große Amiga-2000-Buch" erklärt werden. Nein, das werden wir wirklich nicht.

Doch zurück zu den Diskettenlaufwerken und ihren Namen. Alle Laufwerksbezeichnungen können Sie vor dem Namen eines Inhaltsverzeichnisses eingeben:

```
chdir "df1:Texte"
```

wechselt zum Verzeichnis "Texte" auf der Diskette im Laufwerk 1. Das Ganze ist natürlich nur sinnvoll, wenn das angegebene Laufwerk auch wirklich angeschlossen ist. Wird es von AmigaBASIC nicht gefunden, verlangt Ihr Computer nach einer Diskette mit dem jeweiligen Namen ("Please insert volume df1 in any drive" - "Bitte Diskette namens df1 in ein Laufwerk einlegen"). Aller Wahrscheinlichkeit nach werden Sie keine solche Diskette besitzen. Aber Sie besitzen andere Disketten... Tatsächlich können Sie anstelle von Laufwerksbezeichnungen auch Diskettennamen eingeben:

```
chdir "BASICDisk:Grafik"
```

wechselt zum gewünschten Verzeichnis "Grafik", egal in welchem Laufwerk die Diskette "BASICDisk" eingelegt ist. Ist sie überhaupt nicht eingelegt, bittet Sie der Amiga, die Diskette in eines der Laufwerke zu schieben.

Zu guter Letzt wollen wir Ihnen noch eine Möglichkeit von CHDIR zeigen:

```
chdir "df1:AmigaGalerie/Modernes/Tiere"
```

Das ist die schnellste Methode, an unsere rosa Elefanten zu kommen: Sie können die einzelnen Unter-Verzeichnisse, über die der Weg zum gewünschten Programm führt, hintereinander angeben. Die Namen trennen Sie durch Schrägstriche voneinander ab. Wenn es den von Ihnen angegebenen Weg nicht gibt,

weil Sie zum Beispiel ein Verzeichnis ausgelassen haben, teilt Ihnen AmigaBASIC das durch einen "File not found"-Error mit. Mit ein wenig Übung werden Sie sich perfekt wie ein Klammeraffe (entschuldigen Sie bitte, aber dieser Vergleich paßt so gut zu unseren Bäumen!) durch die Inhaltsverzeichnisse Ihrer Amiga-Disketten hangeln können.

Doch AmigaBASIC bietet noch mehr: Neben dem Laden und Speichern von Programmen oder dem Auswählen und Anzeigen von Inhaltsverzeichnissen können Sie Ihre Programme auch noch umbenennen und löschen. Wir wollen das anhand eines Beispiels durchspielen: Tippen Sie ein paar Zeilen im LIST-Window. Sie brauchen keine Mühe darauf zu verwenden, ein lauffähiges Programm zu erstellen, denn wir wollen das Ganze später sowieso wieder löschen.

Zuerst speichern Sie Ihr Programm (oder wie Sie das Ergebnis sonst bezeichnen möchten) bitte ab. Wählen Sie dann "Save As" aus dem "Project"-Pulldown. Geben Sie als Namen "Test" oder etwas ähnliches an und drücken Sie <RETURN>. Ihr Amiga speichert die Datei auf Diskette. Mit

```
files
```

können Sie das überprüfen. Irgendwo in der Liste tauchen auch die Dateien "Test" und "Test.info" auf.

Doch auf einmal gefällt Ihnen der Name "Test" nicht mehr. Vielleicht sind Sie der Meinung, daß er nicht genug über den Inhalt des Programms aussagt. Der Name "Unwichtig" gefällt Ihnen beispielsweise wesentlich besser. Sie könnten jetzt natürlich das Programm, das ja immer noch im Speicher steht, unter dem neuen Namen abspeichern. Das wäre allerdings Platzverschwendung, schließlich steht dasselbe Programm unter dem Namen "Test" schon einmal auf der Diskette. Stattdessen empfehlen wir Ihnen einen neuen BASIC-Befehl. Er heißt NAME ("to name" bedeutet "benennen, mit einem Namen versehen"). Mit dem NAME-Befehl haben Sie die Möglichkeit, den Namen einer Datei zu verändern. Um "Test" in "Unwichtig" umzutaufen, geben Sie im BASIC-Window ein:

```
name "Test" as "Unwichtig"
```

Das Laufwerk läuft kurz, und schon hat die Datei einen neuen Namen. Sehen Sie mal im Inhaltsverzeichnis nach. Die ehemalige Datei "Test" heißt jetzt wirklich "Unwichtig". Auch die Info-Datei wurde verändert - sie heißt jetzt "Unwichtig.info". Sollte das bei Ihnen nicht so sein (steht also im Inhaltsverzeichnis eine Datei namens "Unwichtig", aber keine Datei "Unwichtig.info", stattdessen immer noch "Test.info"), dann haben Sie eine veraltete Version von AmigaBASIC. Es gab nämlich eine Zeitlang das Problem, daß BASIC-Befehle wie NAME die Info-Datei schlichtweg vergessen haben. In diesem Fall sollten Sie Ihren Händler bitten, für Sie die neueste Version der "ExtrasD"-Diskette zu kopieren. Darauf ist dann ein AmigaBASIC, bei dem dieser Fehler nicht mehr auftritt.

Gehen wir aber noch einen Schritt weiter. Der nächste Befehl ist besonders interessant, weil man mit ihm unwichtige Dateien (wie zum Beispiel unser Programm "Unwichtig") von der Diskette löschen kann. Während Sie auf der Workbench unerwünschte Programme in den Trashcan werfen und dann auch noch den Müll selber ausleeren müssen ("Empty Trash"), geht es in Amiga-BASIC schneller. Das ist oft ein Vorteil, birgt aber auch Gefahren: Überzeugen Sie sich bitte unbedingt vorher, ob eine Datei auch wirklich gelöscht werden darf. Ein Zurück gibt es nämlich nicht mehr. Den BASIC-Befehl KILL sollten Sie also mit der gebührenden Vorsicht behandeln. Nicht umsonst heißt "kill" auf Deutsch "töten, umbringen". Und Sie wollen doch nicht zum Mörder unschuldiger Programme werden?!

Wenn Sie sich nach reiflicher Überlegung entschlossen haben, "Unwichtig" zu löschen, geben Sie bitte ein:

```
KILL "Unwichtig"
```

Kurz und schmerzlos läßt Ihr Amiga das Programm verschwinden. Bei dieser Gelegenheit wollen wir Ihnen einen kleinen Trick verraten: Wenn Sie nur die Info-Datei löschen, erscheint Ihr Programm nicht auf der Workbench, kann aber von Amiga-BASIC ganz normal verarbeitet werden. Da Sie, solange Sie mit

BASIC arbeiten, normalerweise sowieso keine Icons sehen, ist das unter Umständen kein großer Verlust. Es macht im Gegenteil die Inhaltsverzeichnisse, die Sie sich mit FILES anzeigen lassen, übersichtlicher, da nur die eigentlichen Programmnamen erscheinen. Auch zum Schutz Ihrer Programme vor Neugierigen und Unbefugten ist diese Methode ganz hilfreich. Um diesen Effekt zu erreichen, brauchen Sie nur einzugeben:

```
KILL "Unwichtig.info"
```

Anstelle von "Unwichtig" sollten Sie natürlich den Namen der Datei angeben, deren Info-Datei Sie löschen wollen.

Mit dem Repertoire an Disketten-Befehlen, das Sie bisher kennengelernt haben, können Sie jetzt schon in beachtlichem Umfang mit Programmen und Dateien arbeiten. Aber das Interessanteste kommt erst noch: Die Verwaltung von Daten auf Diskette. Damit meinen wir nicht nur trockene Adressenlisten und Aktienkurse, sondern auch die Möglichkeit, Bilder abzuspeichern und vieles mehr. Aber lassen Sie sich überraschen!

3.3 Die Datensammler - ein kleines BASIC-Adreßbuch

Bisher haben wir fast nur Programme auf Diskette abgespeichert. Die beiden einzigen Male, wo etwas anderes vorkam (die Objekt-Daten des ObjectEditor und die Muster des Malprogramms), vertrösteten wir Sie auf später. Um es einfach zu sagen: Jetzt ist "später". Programme sind nämlich bei weitem nicht das einzige, was man auf Disketten finden kann. Auch "Daten" können abgespeichert werden. Diese "Daten" beinhalten die verschiedensten Informationen: Namen, Zahlen, Texte, Bilder und vieles mehr. Die meisten Programme machen auch von der Möglichkeit Gebrauch, ihre Daten auf Diskette abzuspeichern. Denn diesen Informationen ist ja nach dem Ausschalten des Amiga das gleiche Schicksal beschieden wie den Programmen: Sie werden gelöscht. Damit ein Textverarbeitungs-Programm früher geschriebene Texte weiterverwenden, ein Dateiverwaltungs-Programm die eingegeben Informationen darstellen, ein Malprogramm seine Bilder einlesen und ein Musikprogramm

alte Kompositionen neu erklingen lassen kann, werden Dateien auf Diskette abgespeichert und somit gesichert. Sie merken schon: Datenverwaltung ist keineswegs nur ein Thema für Buchhalter und Artverwandte.

Auch AmigaBASIC bietet die Möglichkeit, Daten zu speichern und später wieder einzulesen. Die einfachste Art wollen wir Ihnen jetzt vorstellen. Das folgende Programm legt eine Adreßdatei auf Diskette an.

```
OPEN "Adreßdatei" FOR OUTPUT AS 1

Eingabe:
  PRINT
  INPUT "Name";Nam$
  INPUT "Straße";Stra$
  INPUT "PLZ Wohnort";Ort$
  INPUT "Telefon";Tel$
  PRINT#1,Nam$
  PRINT#1,Stra$
  PRINT#1,Ort$
  PRINT#1,Tel$
  x=x+1
  PRINT x".Datensatz ("Nam$") gespeichert."
  PRINT "Sollen weitere Adressen eingegeben werden?"
  INPUT "J/N:",Antw$
  IF UCASE$(Antw$)="J" THEN Eingabe

CLOSE
PRINT "Datei geschlossen, Eingabe beendet."
```

Die Beispiele zum Thema Dateiverwaltung werden wir in der Schublade "Daten" ansiedeln. Geben Sie also bitte zuerst ein:

```
chdir "BASICDisk:Daten"
```

Speichern Sie das Programm dann unter dem Namen "Adressen schreiben" ab. Bei Programm- oder Dateinamen dürfen auch Leerzeichen vorkommen, AmigaBASIC hat damit keine Pro-

bleme. Wenn Sie das Programm von unserer beigelegten Diskette geladen haben, brauchen Sie's natürlich nicht eigens abzuspeichern. Sie finden es unter dem Namen "Adressen schreiben" in der "Daten"-Schublade unserer Diskette im Buch.

Wie funktioniert unser Beispielprogramm? Das wollen wir uns jetzt Zeile für Zeile ansehen: Der Befehl OPEN in der ersten Zeile erledigt mehrere Aufgaben: Er eröffnet eine Datei namens "Adreßdatei" ("open" heißt ja "öffnen"). Bevor ein BASIC-Programm mit einer Datei arbeiten kann, muß sie vom OPEN-Befehl geöffnet werden. Klar: Solange ein Aktenordner geschlossen ist, können Sie auch nicht lesen, was darin steht. Es sei denn, Sie haben einen Röntgenblick. Bei diesem Befehl wird auch gleich festgelegt, ob in die Datei geschrieben werden soll (OPEN...FOR OUTPUT) oder ob aus der Datei gelesen wird (OPEN...FOR INPUT). Nur eine dieser beiden Funktionen kann auf einmal mit derselben Datei durchgeführt werden. Allerdings ist es möglich, mehrere Dateien zu öffnen, von denen dann einige "Input"- und andere "Output"-Dateien sind.

Aus diesem Grund wird jeder offenen Datei eine Nummer zugeteilt (in unserem Fall die Nummer 1), damit bei Lese- und Schreib-Befehlen angegeben werden kann, welche Datei gemeint ist. Wenn, wie in unserem Fall, eine Output-Datei zum ersten Mal eröffnet wird, sorgt OPEN dafür, daß sie auf der Diskette neu angelegt wird. Die Datei entsteht immer in der aktuellen Schublade (bzw. im aktuellen Inhaltsverzeichnis). Die Befehlssyntax sieht also so aus:

OPEN "(Dateiname)" FOR (Modus) AS (Dateinummer)

Es gibt noch eine zweite Syntax. Sie beinhaltet die gleichen Angaben, aber in anderer Reihenfolge:

OPEN "(Modus)", #(Dateinummer), "(Dateiname)"

Wir hätten also in der ersten Zeile auch schreiben können:

```
OPEN "O",#1,"Adreßdatei"
```

Welche der beiden Schreibweisen Sie verwenden, ist in diesem Fall egal. Je nach den Umständen kann die eine oder die andere Art zweckmäßiger sein, wir kommen noch darauf zurück.

Bisher haben wir eine Datei auf Diskette erzeugt, die "Adreßdatei" heißt, in die wir Daten schreiben werden und die die Nummer 1 hat. Nun sorgen wir dafür, daß es auch Daten gibt, die wir abspeichern können.

Zu Beginn des Teils 'Eingabe:' fragen wir den Anwender mit INPUT-Befehlen nach den Inhalten der Variablen 'Nam$', 'Stra$', 'Ort$' und 'Tel$'. In diesen vier Strings stehen die Angaben, aus denen sich die jeweilige Adresse zusammensetzt. Um diese Daten in die Adreßdatei zu schreiben, verwenden wir den Befehl PRINT#. Das Zeichen # ist im amerikanischen Sprachraum ein Symbol für "Nummer". Bei uns wird es meist "Doppelkreuz" genannt. Wenn Sie "PRINT#" sagen wollen, sagen Sie aber am besten "print file" (Drucke in Datei). Da laufen Sie am wenigsten Gefahr, sich die Zunge zu brechen. Hinter PRINT# steht die Nummer der Datei, auf die sich der Befehl bezieht. Der Befehl PRINT#1,Nam$ bewirkt, daß der Inhalt der Variablen 'Nam$' in die Datei Nummer 1, also unsere Datei "Adreßdatei" geschrieben wird. PRINT# wirkt wie PRINT, nur wird die Ausgabe nicht auf den Bildschirm, sondern in eine Datei geschrieben.

Wir speichern die Adressen in einer sequentiellen Datei. "Sequentiell" bedeutet "hintereinander". Bei einer sequentiellen Datei werden die einzelnen Daten hintereinander abgespeichert. In dem Eintrag auf der Diskette steht zuerst der Name aus der ersten Adresse, dann die Straße, dann der Wohnort und dann die Telefonnummer. Dahinter folgt der zweite Name, die zweite Straße usw. Außer der sequentiellen gibt es auch noch die relative (wahlfreie) Datenspeicherung. Zu ihr kommen wir später.

Der Rest des Programms birgt nicht mehr allzu viel Neues oder Schwieriges für Sie. Die Variable x wird hochgezählt, um auf dem Bildschirm ausgeben zu können, wieviele Adressen bereits

eingegeben wurden. Dann werden Sie gefragt, ob Sie weitere Adressen eingeben wollen. Wenn Sie das mit "J" beantworten, springt das Programm zurück zum Teil 'Eingabe:'.

Eines hätten Sie vielleicht fast übersehen oder für einen Druckfehler gehalten: Beachten Sie den Unterschied zwischen

INPUT "(Text)";Variable

(mit Strichpunkt) und

INPUT "(Text)",Variable

(mit Komma). Die beiden Varianten der INPUT-Befehls kommen in oberen bzw. unteren Teil des Programms vor. Steht ein Stichpunkt zwischen dem Text und der Variablen, erzeugt AmigaBASIC bei der Abfrage auf dem Bildschirm ein Fragezeichen. Verwenden Sie ein Komma, wird das Fragezeichen unterdrückt. Bei der Eingabe von "J" oder "N" wollten wir kein Fragezeichen.

Wenn Sie die Frage, ob weitere Daten folgen, verneinen, indem Sie "N" oder irgendein anderes Zeichen eingeben, führt das Programm den Befehl CLOSE 1 aus. CLOSE schließt eine Datei, die mit OPEN geöffnet wurde. Dabei passiert im einzelnen folgendes:

Bei der Speicherung auf Diskette wird nicht jedes einzelne Zeichen direkt auf die Diskette geschrieben. Das würde viel zu lange dauern und das Diskettenlaufwerk unnötig besetzen. Gerade durch das Multitasking auf dem Amiga wollen ja vielleicht auch noch andere Programme auf die Diskette zugreifen. Sie wären blockiert, solange von AmigaBASIC eine oder mehrere Dateien geöffnet sind. Um das zu vermeiden, wird für jede offene Datei ein Puffer angelegt. Das ist ein Speicherbereich, normalerweise 128 Bytes groß, in dem die Zeichen solange gesammelt werden, bis er voll ist. Dann werden 128 Zeichen auf einmal auf die Diskette geschrieben. CLOSE sorgt dafür, daß der Pufferinhalt abgespeichert wird, obwohl der Puffer noch nicht voll ist.

Als nächstes wird der Eintrag auf Diskette abgeschlossen. Solange eine Datei zum Lesen oder Schreiben geöffnet ist, kann sie nicht erneut geöffnet werden. Sie kann auch nicht mit KILL gelöscht oder mit NAME umbenannt werden. Um alle diese Funktionen wieder zu ermöglichen, muß die Datei offiziell beendet werden. Auch dafür sorgt CLOSE. Außerdem gibt CLOSE die Dateinummer wieder frei, sie darf für eine neue Datei verwendet werden.

Folgen nach CLOSE noch Lese- oder Schreibzugriffe auf die angegebene Datei, ist ein "Bad file number"-Error die Folge. Sie sehen, wieviel hinter so einem unscheinbaren Befehlchen wie CLOSE stecken kann. Die Bedienung des Programms dürfte Ihnen keine Schwierigkeiten bereiten: Der Computer fragt Sie Stück für Stück nach den Bestandteilen der Adressen. Ist die Adresse eingegeben, müssen Sie "J" eingeben, um weiterzumachen.

Geben Sie doch einmal die Adressen einiger Ihrer Freunde und Bekannten ein oder auch die Adressen Ihres Finanzamts, Ihrer Kommunalverwaltung und Ihres Arbeitgebers. Dabei brauchen Sie keine Skrupel zu haben, denn diese Institutionen machen mit Ihren Daten genau dasselbe, nämlich sammeln.

Von der Tatsache, daß die Adressen jetzt auf der Diskette stehen, haben wir allerdings noch keinen allzugroßen Nutzen. Wir brauchen ein Programm, das die Adressen aus der Datei wieder ausliest. Das nächste Programm dient dazu. Bevor Sie es eingeben, speichern Sie bitte das "Adressen schreiben"-Programm ab, falls Sie es bisher noch nicht getan haben. Löschen Sie dann den Speicher mit NEW und tippen Sie das Listing ab oder laden Sie es von der beiliegenden Diskette. (Es ist das Programm "Adressen lesen" in der "Daten"-Schublade.)

```
OPEN "Adreßdatei" FOR INPUT AS 1

Einlesen:
  INPUT#1,Nam$
  INPUT#1,Stra$
```

```
    INPUT#1,Ort$
    INPUT#1,Tel$

    PRINT
    PRINT "Name: ";Nam$
    PRINT "Straße: ";Stra$
    PRINT "PLZ Wohnort: ";Ort$
    PRINT "Telefon: ";Tel$

    IF EOF(1)=0 THEN Einlesen
  CLOSE 1
```

Wenn Sie das Programm starten, sehen Sie, was passiert: Die einzelnen Adressen aus der Datei werden gelesen und auf dem Bildschirm ausgegeben. Speichern Sie das Programm bitte unter dem Namen "Adressen lesen", wenn Sie es selbst eingegeben haben.

Der OPEN-Befehl öffnet die Datei "Adreßdatei" diesmal zum Lesen (FOR INPUT). Das kennen Sie schon, unter anderem auch vom Einlesen der Objekt-Daten aus dem Grafik-Kapitel. Um Daten wirklich wieder einlesen zu können, muß AmigaBASIC die angegebene Datei natürlich auch finden. Anderenfalls gibt es einen "File not found"-Error. Wenn dieser Fehler auftritt und Sie sicher sind, daß der angegebene Name stimmt, müssen Sie möglicherweise das aktuelle Inhaltsverzeichnis mit CHDIR ändern.

So wie wir die Daten mit PRINT# in eine Datei geschrieben haben, können wir sie mit INPUT# wieder auslesen. Wichtig ist dabei, daß wir beim Lesen die Reihenfolge einhalten, in der die Werte in der Datei stehen. Sonst finden wir z.B. einen Herrn "6000 Frankfurt", der in "069/982314" in der Straße "Winfried Müller" wohnt. INPUT# weist die Eingaben einer Variablen zu, genauso wie INPUT. Der einzige Unterschied: Diesmal kommen die Daten nicht von der Tastatur, sondern aus einer Disketten-Datei.

Dann stellt unser Programm die gelesenen Daten auf dem Bildschirm dar. Die Funktion EOF (End of File - Ende der Datei) gibt Auskunft darüber, ob sich noch weitere Daten in der Datei befinden. In Klammern geben Sie die Dateinummer an. Das Ergebnis ist 0, solange noch Daten vorhanden sind. Wurde das letzte Zeichen gelesen, ergibt EOF(Dateinummer) den Wert -1. So können Sie bei jedem Durchgang feststellen, ob weiter gelesen oder die Datei geschlossen werden soll. Falls Sie nämlich versuchen, Werte aus einer Datei zu lesen, aus der schon alle Werte gelesen wurden, erhalten sie einen "Input past End"-Error. Sie haben also versucht, hinter dem Dateiende Werte einzulesen.

Das Lesen aus sequentiellen Dateien funktioniert ähnlich wie die Befehle READ und DATA: Mit jedem Lesen wird ein Zeiger um eine Position weiter nach oben gesetzt. So erhalten Sie immer den nächsten noch nicht gelesenen Wert. Wie gesagt, das Programm schreibt die Adressen in der Reihenfolge auf den Bildschirm, in der sie in der Datei stehen. Das Hauptmerkmal von sequentiellen Dateien ist, daß ein Datensatz nach dem anderen gelesen werden muß. Wenn Sie z.B. die vierte Adresse suchen, müssen die ersten drei trotzdem vorher eingelesen werden. Es gibt keine Möglichkeit, sie zu überspringen. Sequentielle Dateien eignen sich daher besonders gut für Listen, Verzeichnisse oder Daten, die ständig fortgeschrieben werden (z.B. Kursentwicklungen).

Es wäre ziemlich umständlich, wenn in dem Augenblick, wo eine Datei um neue Daten erweitert werden soll, erst einmal alle alten Daten eingelesen und neu abgespeichert werden müßten. Dagegen gibt es aber glücklicherweise ein gutes Mittel: Dateien können nämlich nicht nur zum Schreiben (FOR OUTPUT) und zum Lesen (FOR INPUT) geöffnet werden, sondern auch zum Anhängen neuer Daten. Dieser Modus heißt APPEND (engl. append = anfügen). Das können Sie ganz einfach ausprobieren. Laden Sie "Adressen schreiben", das Programm, mit dem Sie Adressen eingeben können. Ändern Sie die erste Zeile, indem Sie anstelle von OUTPUT das Wort APPEND angeben:

```
OPEN "Adreßdatei" FOR APPEND AS 1
```

Starten Sie das Programm und geben Sie noch ein oder zwei Adressen ein. Stören Sie sich bitte nicht daran, daß das Programm der Meinung ist, es schreibe den 1. und 2. Datensatz - die Nummern beziehen sich auf die neuen, angefügten Daten.

Nach Beendigung der Datenerfassung (klingt professionell, nicht wahr?) laden Sie bitte wieder "Adressen lesen" und starten dieses Programm. Sie werden feststellen, daß die neuen Adressen hinter den alten Eintragungen angehängt wurden. Auch das ist ein wichtiger Punkt bei sequentiellen Dateien: Sie können jederzeit Daten anhängen. Dagegen ist es nicht möglich, Daten in der Mitte oder am Anfang der Datei einzufügen, zu verändern oder zu löschen.

Wenn Sie wollen, können Sie Dateien grundsätzlich im APPEND-Modus zum Schreiben öffnen: Sollte es die angegebene Datei bisher noch nicht geben, verhält sich AmigaBASIC genauso wie im OUTPUT-Modus: Es erzeugt eine neue Datei.

Unsere Kenntnisse über sequentielle Dateien wollen wir jetzt in einem etwas größeren Programm verwenden. Im Grafik-Teil dieses Buchs haben wir Ihnen ein Utility versprochen, das Ihnen hilft, die Daten für Ihre Balken- und Tortengrafiken zu verwalten. Dieses Programm stellen wir Ihnen im nächsten Kapitel vor.

3.4 Woher kommen die kleinen Balken und Torten? Die Statistikdaten-Verwaltung

Irgendwoher müssen Balken- und Tortengrafiken ja ihre Daten beziehen. Bisher haben Sie die immer von Hand eingegeben, aber das ist nicht eben der Weisheit letzter Schluß. Angenommen, Sie wollen die Aktienkurse von... (haben Sie etwa gedacht, wir machen hier Schleichwerbung?!) der Firma XY grafisch darstellen. Spätestens nach 14 Tagen würde es Ihnen wohl ziemlich auf den Wecker gehen, dazu jedesmal alle Kurse nochmals abzutippen. Eben deshalb schreiben wir jetzt, mit unserem Wis-

sen um die Verwaltung von Daten und Fakten auf dem Amiga, ein kleines Programm, das sich solche Zahlen merkt und dem Sie jeden Tag die aktuellen Kurse bedenkenlos anvertrauen können.

Das Programm, das aus statistischen Daten Balken- und Tortengrafiken erzeugt, besitzen Sie ja schon. Es befindet sich wahrscheinlich in der "Grafik"-Schublade auf Ihrer BASIC-Diskette. Oder ganz sicher in der "Grafik"-Schublade auf unserer beigelegten Diskette.

Wenn Sie unser Adressen-Programm von vorhin selbst eingetippt haben, haben Sie ja sicher schon lange abgespeichert. Löschen Sie also nun bitte den Speicher mit NEW. Laden Sie dann die Version des Balken- und Tortengrafik-Utilities, die ohne Testeingabe-Teil abgespeichert wurde. Wahrscheinlich müssen Sie dabei folgende Arbeitsschritte durchführen:

- Wechseln Sie ins Grafik-Inhaltsverzeichnis mit CHDIR "BASICDisk:Grafik".

- Suchen Sie den Namen der gewünschten Datei, indem Sie mit FILES das Inhaltsverzeichnis anzeigen lassen.

- Laden Sie das Programm. Das könnte zum Beispiel so aussehen: LOAD "BalkenTorten".

- Wechseln Sie jetzt bitte ins Daten-Inhaltsverzeichnis: CHDIR ":Daten".

Jetzt kommt der Schritt, wegen dem wir Sie um die ganze Prozedur gebeten haben: Da wir später den Balken-/Torten-Teil mit unserem neuen Programm MERGEn wollen, müssen Sie ihn bitte als ASCII-Datei abspeichern:

```
SAVE "BalkenTorten",A
```

Das war schon alles, was an vorbereitenden Arbeiten anfällt. Sie können diese Prozedur aber auch bleiben lassen und sich stattdessen auf unsere Diskette im Buch verlassen. Sie finden das Er-

gebnis der nächsten Eingaben unter dem Namen "BTDaten" in der "Daten"-Schublade der beigelegten Diskette.

Für alle, die selber tippen wollen oder sich für die Programmerklärungen interessieren, folgt hier der erste Teil des neuen Programms:

```
Vorbereitungen:
  DIM Nummer$(58),Bez$(58),Zahl$(58)
  DIM Wert$(50),Wert(50)
  FOR x=1 TO 58
    IF x>4 AND x<55 THEN Nummer$(x)=STR$(x-4)
  NEXT x
  HoechstZeile=1

  Farben=3
  SCREEN 4,640,200,Farben,2
  WINDOW 99,"Grafik",,20,4
  PALETTE 7,.8,.2,.1

  WINDOW 1,"Statistik-Daten-Verwaltung",(0,12)-(631,111),22,-1

  MENU 1,0,1,"Datei"
  MENU 1,1,1,"laden"
  MENU 1,2,1,"speichern"
  MENU 1,3,1,"löschen"
  MENU 1,4,1,"Programm beenden"
  MENU 2,0,1,"Grafik"
  MENU 2,1,1,"Balkengrafik"
  MENU 2,2,1,"Tortengrafik"
  MENU 3,0,0,""
  MENU 4,0,0,""

  ON MENU GOSUB Menukontrolle
  MENU ON

GOTO Hauptschleife
```

Wie so oft beginnt unser Programm wieder mit einem Vorbereitungsteil. Hier werden zunächst die Datenfelder dimensioniert, die wir im Verlauf des Programms benötigen werden. Die Namen sprechen fast für sich selbst: 'Nummer$' beinhaltet die laufende Nummer der Datenzeilen und 'Bez$' die Bezeichnung eines Werts. Also z.B. die Namen der Parteien, über deren Glück und Unglück Sie in Ihrer persönlichen Wahl-Hochrechnung entscheiden können. Übrigens: Der FDP würden wir..., aber lassen wir das.

'Zahl$' beinhaltet die zugehörigen Zahlen, also beispielsweise die prozentualen Anteile. Wundern Sie sich nicht darüber, daß wir ein Stringfeld für Zahlen verwenden: Das macht die Eingaberoutine einfacher, da Sie zunächst eingeben können, was Sie wollen (Zahlen, Buchstaben, Satzzeichen usw.). Erst beim Aufruf der Grafik-Routinen werden die Inhalte des Feldes in Zahlen umgerechnet.

'Wert$' und 'Wert' kennen Sie sicher noch vom Balken-/Tortengrafik-Utility: In diesen Feldern übergeben wir die Daten dem Unterprogramm.

Das Feld 'Nummer$' wird mit festem Inhalt versehen. Wie schon erklärt, stehen hier die laufenden Nummern der Datenzeilen. Wir erlauben maximal 50 Werte, die Numerierung wird also von 1 bis 50 gehen. Warum dimensionieren wir das Feld dann aber auf 58 Elemente? Ist das vielleicht ein Sicherheitszuschlag? Ganz falsch liegen Sie mit dieser Vermutung nicht. Aber lassen Sie sich's erklären:

Das Grundprinzip unseres Programms ist, daß Sie immer einen Ausschnitt aus den insgesamt 50 Datenzeilen sehen. Alle 50 würden ja kaum gleichzeitig auf den Bildschirm passen. Eine Datenzeile enthält drei Elemente: Ganz am Anfang die laufende Nummer, dann die Bezeichnung zum Datenwert und am Schluß die jeweilige Zahl.

Sie sehen immer 9 Datenzeilen auf dem Bildschirm. Ihre Arbeiten (Eingabe, Korrektur, Einfügen, Löschen etc.) finden immer in der fünften Zeile statt, also in der goldenen Mitte. Die Zeilen

1 bis 4 und 6 bis 9 dienen nur der Orientierung bzw. dem Vergleich mit den Werten in der näheren Umgebung der bearbeiteten Zeile. Mit den Cursortasten können Sie die Zeilen aufwärts oder abwärts bewegen und sich so die Zeile aussuchen, die Sie bearbeiten wollen. Der Cursor bleibt fest in der mittleren Zeile stehen.

Da Sie natürlich auch die erste und die letzte Zeile der Datei bearbeiten wollen, haben die Datenfelder als Vor- und als Nachspann jeweils vier leere Elemente. Die erste wirkliche Datenzeile befindet sich in den Feldern auf der fünften Position. So erklärt sich die Zahl 58. Denn 4 + 50 + 4 ergibt...? Eben. Die FOR...NEXT-Schleife erzeugt im Feld 'Nummer$' die Numerierung, wie wir sie beschrieben haben: Vier Positionen bleiben leer, dann folgen die Zahlen von 1 bis 50, dann wieder vier Leerpositionen. Mit dem Befehl STR$ können Sie Zahlen ins Strings umwandeln. Das ist die Gegenfunktion zu dem Ihnen schon bekannten VAL.

```
? VAL(STR$(5))
```

ergibt wieder die Zahl 5. Das Umwandeln bestimmter Datenformate in andere ist so beliebt, daß Sie zu diesem Thema bald noch mehr Befehle kennenlernen werden.

Die Variable 'HoechstZeile' wird auf 1 gesetzt. In dieser Variablen steht, bis zu welcher Zeile Daten zu finden sind. Da am Anfang noch überhaupt keine Daten in den Feldern stehen, ist die Zeile 1 die erste und gleichzeitig die letzte Zeile.

Über 'Farben' müssen wir sicher keine Worte mehr verlieren. Sie können nach Belieben ein bis vier Bit-Ebenen angeben, je nachdem wieviel Farbe Sie in Ihren Statistiken bevorzugen.

Dann erzeugen wir Screen und Window für das Grafik-Unterprogramm und definieren die Farbe Nummer 7 von Weiß auf Rot um. Das kommt Ihnen bekannt vor? Natürlich, diese Vorbereitungen kennen Sie noch von den Beschreibungen zum Balken-/Tortengrafik-Utility aus Kapitel 2.7.

Window 1 (das BASIC-Window) wird etwas verkleinert und bekommt den Namen "Statistikdaten-Verwaltung". Wenn Ihnen das zu lang oder zu trocken ist und Ihnen etwas besseres einfällt, können Sie gern einen anderen Namen eingeben. Als Screen haben wir -1 angegeben. Während die Screens 1 bis 4 allein dem Anwender zur Verfügung stehen, ist -1 die Nummer des Workbench-Screens.

Es folgt die Definition der Pulldown-Menüs. Denn auf Menüsteuerung wollen wir in diesem Programm nicht verzichten: In einem Pulldown namens "Datei" stehen Ihnen die Optionen "laden" und "speichern", außerdem "löschen" (gemeint: Löschen der Daten im Speicher) und "Programm beenden" zur Verfügung.

Das Pulldown "Grafik" dient zum Aufruf der beiden Möglichkeiten zur Aufbereitung statistischer Daten: Balken- oder Tortengrafiken. Die Menüs 3 und 4, die in der BASIC-Grundeinstellung ja mit "Run" und "Windows" belegt sind, werden unterdrückt.

Schließlich aktivieren wir Event Trapping für die Menü-Steuerung und springen zur 'Hauptschleife:'.

Im Gegensatz zum Malprogramm ist dieses Programm nicht ausschließlich durch Event Trapping gesteuert. Vielmehr sind die Pulldown-Funktionen mehr oder weniger willkommene Unterbrechungen der Programmteile 'Hauptschleife:' und 'Eingabe:'. Diese Handhabung des Event Trapping ist die üblichere Methode. Sie ist immer dann sinnvoll, wenn die Reaktionen auf mögliche Ereignisse (Menüs, Maus, Timerabläufe, Fehlerbehandlung, Abbruchkontrolle sowie Kollisionsabfrage von Grafikobjekten) das Hauptprogramm ergänzen oder erweitern.

Als nächstes erwartet unser Programm zu Recht einige Informationen darüber, was es bei einer Menü-Auswahl überhaupt tun soll:

```
Menukontrolle:
  Men=MENU(0) : Menpunkt=MENU(1)
  IF Men=1 THEN
    IF Menpunkt=1 THEN GOSUB DatenLaden
    IF Menpunkt=2 THEN GOSUB DatenSpeichern
    IF Menpunkt=3 THEN GOSUB DatenLoeschen
    IF Menpunkt=4 THEN Ende
  END IF
  IF Men=2 THEN
    IF Menpunkt=1 THEN Wert$(0)="B"
    IF Menpunkt=2 THEN Wert$(0)="T"
    Wert(0)=HoechstZeile
    IF Zahl$(Wert(0)+4)="" THEN Wert(0)=Wert(0)-1
    FOR x=1 TO Wert(0)
      Wert$(x)=Bez$(x+4)
      Wert(x)=VAL(Zahl$(x+4))
      IF Wert(x)=0 THEN Wert(x)=.01
    NEXT x
    MENU OFF
    MENU 1,0,0 : MENU 2,0,0
    WINDOW 99 : CLS

    GOSUB Grafik

    WINDOW 2,"Bitte drücken Sie eine Taste!",(350,0)-(631,0),20,4
    COLOR 0,1 : CLS
    WHILE INKEY$=""
    WEND
    WINDOW CLOSE 2
    WINDOW 1
    MENU ON
    MENU 1,0,1 : MENU 2,0,1
  END IF
RETURN
```

In der Variablen 'Men' legen wir die Nummer des gewählten Pulldowns ab, in 'Menpunkt' die Nummer der gewählten Menü-Option. Genauso haben wir's übrigens auch im Malprogramm

gehandhabt. Sie sehen: Langsam wiederholen sich immer mehr dieselben Prozeduren, wenn es darum geht, Probleme zu lösen.

Falls das erste Pulldown gewählt wurde, ist für jeden möglichen Menüpunkt ein eigenes Unterprogramm zuständig. Wurde das zweite Pulldown gewählt, soll eine der beiden Grafikarten aufgerufen werden. In der ersten Position des Felds 'Wert$' dient ein B (für Balken) oder ein T (für Torten) zur Kennzeichnung, welche Darstellung gewünscht wird.

Die erste Position von 'Wert' muß beim Aufruf des 'Grafik'-Unterprogramms die Anzahl an Daten enthalten. Die Variable 'HoechstZeile" beinhaltet diesen Wert, sie kann einfach übernommen werden. Allerdings schießt 'HoechstZeile' manchmal etwas übers Ziel hinaus: Wenn nämlich nach der letzten Dateneingabe die <RETURN>-Taste gedrückt wurde, steht der Cursor schon eine Zeile unterhalb der letzten Datenzeile. In diesem Fall würde sich ein Wert 0 ohne Bezeichnung in die Grafiken einschleichen, was unsinnig wäre. Deshalb prüft das Programm, ob in der Zahlen-Spalte der letzten Zeile auch eine Zahl steht. Ist das nicht der Fall, handelt es sich um eine Leerzeile und der Wert in 'Wert(0)' wird um 1 vermindert.

Die nun folgende FOR...NEXT-Schleife kopiert die Inhalte der Felder, aus denen sich die Datenzeilen zusammensetzen, in die Übergabefelder 'Wert' und 'Wert$'. Erinnern Sie sich daran, daß die Felder 'Bez$' und 'Zahl$' um vier Postionen "vorgehen". (Deshalb: Wert$(x)=Bez$(x+4)) Die Schleife wird so oft durchgeführt, wie Daten vorhanden sind.

An dieser Stelle rechnet die Funktion VAL auch die Inhalte des String-Felds 'Zahl$' in Zahlen für das Zahlen-Feld 'Wert' um. Kommt unter den Daten eine 0 vor, wird der Wert etwas erhöht, nämlich auf 0,01. Die Grafikroutinen würden sonst einen "Division by Zero"-Error verursachen.

Während des Zeichnens soll die Menüauswahl abgeschaltet werden, deshalb MENU OFF. Die beiden Pulldown-Menüs erscheinen nur noch in Geisterschrift, um auch optisch klarzumachen, daß zur Zeit keine Auswahl möglich ist. WINDOW 99 wird akti-

viert. Das ist das Window, in dem die Grafik entsteht. Vorher löschen wir es noch mit CLS, denn ab dem zweiten Aufruf befindet sich schon eine frühere Grafik im Window.

Der Rest ist Sache des 'Grafik:'-Unterprogramms, das jetzt mit GOSUB aufgerufen wird.

Nach der Fertigstellung der Grafik kehrt das Programm zur Zeile des Aufrufs zurück. Wir erzeugen jetzt ein Window, das in der rechten oberen Bildschirmecke erscheinen und den Titel tragen wird: "Bitte drücken Sie eine Taste!". Das eigentliche Window ist nur zwei Pixels hoch, denn im Titel ist schon alles Wichtige gesagt. Auch das ist eine Möglichkeit, wenn Sie Dialog-Windows verwenden wollen. Falls Sie für *Hardcopies* (Ausdrucke auf einem Drucker) oder Fotos die Grafik pur brauchen, können Sie das Dialog-Window in den Hintergrund klicken.

Eine Hardcopy, die den Bildschirminhalt je nach den Fähigkeiten Ihres Druckers in Farbe oder durch Umsetzung der Farben in Grautöne zu Papier bringt, bietet Ihnen das Programm "GraphicDump", das Sie in der "System"-Schublade der Workbench-Diskette finden. AmigaBASIC kann von sich aus keine Hardcopies drucken.

Zur Funktionsweise von "GraphicDump" sollten Sie vor allem eines wissen: Ausgedruckt wird immer der vordere Screen. Solange das BASIC-Window auf dem Workbench-Screen dargestellt wird, gibt es keine Probleme. Wollen Sie jedoch den Inhalt eines anderen Screens ausdrucken, sollten Sie den gewünschten Screen nach unten ziehen (das macht "GraphicDump" nichts aus), bis der Workbench-Screen erscheint, auf dem Sie dann das "GraphicDump"-Icon anklicken können. Wenn Sie diesen Hinweis nicht beachten, bekommen Sie ausschließlich Hardcopies des Workbench-Screens, weil der beim Anklicken von "GraphicDump" der vordere Screen war.

Doch zurück zu unserem Programm: Eine WHILE...WEND-Schleife wartet nun auf den Tastendruck. Fand dieses weltbewegende Ereignis schließlich statt, wird zunächst unser Dialog-Window geschlossen, dann Window 1 (das BASIC- bzw. Daten-

eingabe-Window) aktiviert. Schließlich lassen wir Event Trapping für die Menü-Auswahl zu, aktivieren die beiden Pulldowns und kehren mit RETURN zu der Programmzeile zurück, mit der unser Amiga gerade beschäftigt war, als er durch das Menü-Ereignis unterbrochen wurde - was eine kleine Maus alles bewirken kann...

Leider können Sie die bisher eingegebenen Teile noch nicht ausprobieren, denn das 'Grafik:'-Unterprogramm fehlt noch, und auch mit Daten irgendwelcher Art können wir bisher noch nicht dienen. Gehen wir deshalb mal wieder ein Stück Programm an:

```
Hauptschleife:
  CLS
  IF HoechstZeile>50 THEN HoechstZeile=50
  IF Zeile1>HoechstZeile THEN Zeile1=HoechstZeile : BEEP
  IF Zeile1<1 THEN Zeile1=1 : BEEP
  PRINT "Nummer";TAB(10);"Bezeichnung";TAB(45);"Wert"
  FOR x=Zeile1 TO Zeile1+8
    COLOR 1,0
    PRINT Nummer$(x);TAB(10);Bez$(x);TAB(45);Zahl$(x)
  NEXT x
  IF BezWert=0 THEN StartSpalte=10 : EndSpalte=40
  IF BezWert=1 THEN StartSpalte=45 : EndSpalte=55
  xp=StartSpalte

  GOSUB Eingabe
  in$=""

GOTO Hauptschleife


Eingabe:
  IF xp<StartSpalte THEN xp=StartSpalte
  LOCATE 6,xp
  COLOR 0,3 : PRINT " "; : COLOR 1,0
  i$=INKEY$
  IF i$="" THEN Eingabe
  IF i$=CHR$(1) THEN Zeile1=1 : RETURN
  IF i$=CHR$(5) THEN Zeile1=HoechstZeile : RETURN
```

```
  IF i$=CHR$(12) THEN Loeschen: RETURN
  IF i$=CHR$(14) THEN Einfuegen : RETURN
  IF i$=CHR$(28) THEN GOSUB Textuebernahme : xp=StartSpalte : Zeile1=Zei
le1-1 : RETURN
  IF i$=CHR$(29) THEN GOSUB Textuebernahme : xp=StartSpalte : Zeile1=Zei
le1+1 : RETURN

  TextPos=xp-StartSpalte+1
  IF BezWert=0 THEN Text$=Bez$(Zeile1+4)
  IF BezWert=1 THEN Text$=Zahl$(Zeile1+4)

  IF i$=CHR$(30) THEN
    IF TextPos<=LEN(Text$) THEN i$=MID$(Text$,TexPos,1)
  END IF

  IF i$=CHR$(13) OR i$=CHR$(9) THEN
    GOSUB Textuebernahme
    BezWert=1-BezWert
    IF BezWert=0 THEN Zeile1=Zeile1+1
    xp=StartSpalte
    IF HoechstZeile<Zeile1 THEN HoechstZeile=Zeile1
    RETURN
  END IF
  IF i$=CHR$(8) OR I$=CHR$(31) THEN
    LOCATE 6,xp
    IF TextPos<=LEN(Text$) THEN
      PRINT RIGHT$(Text$,LEN(Text$)-TextPos+1);
    ELSE
      PRINT " ";
    END IF
    xp=xp-1 : IF xp<StartSpalte THEN xp=StartSpalte : BEEP : GOTO Eingabe
    in$=LEFT$(in$,(LEN(in$)-1))
    GOTO Eingabe
  END IF
  IF i$=CHR$(34) THEN i$=CHR$(39)
  IF i$>CHR$(31) AND i$<CHR$(127) THEN
    IF xp>=EndSpalte THEN xp=EndSpalte : BEEP : GOTO Eingabe
    LOCATE 6,xp
    PRINT i$;
```

```
    in$=in$+i$
    xp=xp+1
  END IF
GOTO Eingabe

Textuebernahme:
  IF in$<>"" THEN
    IF BezWert=0 THEN Bez$(Zeile1+4)=in$
    IF BezWert=1 THEN Zahl$(Zeile1+4)=in$
    in$=""
    AltDaten=1
  END IF
RETURN

Loeschen:
  FOR x=Zeile1+4 TO 54
    Bez$(x)=Bez$(x+1)
    Zahl$(x)=Zahl$(x+1)
  NEXT x
  HoechstZeile=HoechstZeile-1
  IF HoechstZeile<1 THEN HoechstZeile=1
RETURN

Einfuegen:
  IF HoechstZeile>=50 THEN BEEP : RETURN
  FOR x=HoechstZeile+4 TO Zeile1+4 STEP -1
    Bez$(x+1)=Bez$(x)
    Zahl$(x+1)=Zahl$(x)
  NEXT x
  Bez$(Zeile1+4)=""
  Zahl$(Zeile1+4)=""
  HoechstZeile=HoechstZeile+1
RETURN
```

Die Aufgabenverteilung im Programm sieht so aus: 'Eingabe:' ist für das Eintippen der Daten zuständig und 'Hauptschleife:' für das Scrolling der Zeilen.

Oh Verzeihung! *Scrolling* ist ja auch wieder so ein Fachausdruck. Er setzt sich zusammen aus den englischen Begriffen "Screen" (Bildschirm) und "rolling" (Rollen). Von Scrolling, also Bildschirmrollen, spricht man, wenn irgendwelche Zeilen, Zeichen oder ähnliche Daten an einer Seite des Bildschirms verschwinden und dafür auf der gegenüberliegenden Seite neue Daten in den Bildschirm wandern. Scrolling ist die sinnvollste Methode, Daten, die nicht auf einmal auf den Bildschirm passen, sichtbar zu machen. Der Anwender sieht immer einen Ausschnitt aus der Gesamtdarstellung.

Mit jeder Verschiebung des Bildschirminhalts baut die 'Hauptschleife:' die Textdarstellung neu auf. Zuerst löscht CLS den Bildschirm. Der Wert von 'HoechstZeile' wird auf 50 begrenzt, mehr Zeilen sind nicht zugelassen.

Die Variable 'Zeile1' gibt die Nummer der ersten Zeile auf dem Bildschirm an. Genauer gesagt, beinhaltet 'Zeile1' die Feldposition, an der die verschiedenen Daten der ersten von neun dargestellten Zeilen gespeichert sind. Lesen Sie den letzten Satz ruhig noch mal, wenn Sie ihn nicht auf Anhieb verstanden haben. Ein Beispiel soll verdeutlichen, was gemeint ist: Hat 'Zeile1' den Wert 1, werden in der ersten Zeile die Werte aus 'Nummer$(1)', 'Bez$(1)' und 'Zahl$(1)' dargestellt. Wie Sie sicher noch wissen, sind das alles leere Daten, die erste Zeile ist also leer. Die fünfte Zeile ist die erste Zeile, die vom Cursor erreicht werden kann. Sie hat deshalb die laufende Nummer 1. In 'Nummer$(5)' steht die Zahl 1. 'Zeile1' gibt also gleichzeitig auch die laufende Nummer der aktuellen Eingabezeile an.

Vermutlich haben Sie noch immer kräftige Schwierigkeiten, dem Ganzen zu folgen. Aber keine Sorge: Alle Theorie ist grau - besser verständlich ist das alles, wenn Sie das Programm mal ausprobieren. Sie haben jetzt einen so großen Teil hinter sich gebracht, daß das Programm in seinen Grundfunktionen schon läuft.

Vergessen Sie aber bitte vorher das Abspeichern nicht. Sollten Fehler auftreten, korrigieren Sie sie bitte anhand unseres Listings. Wenn alles richtig läuft, drücken Sie bitte ein paarmal

<RETURN>. Damit machen Sie leere Eingaben, aber das ist im Moment nicht so wichtig. Sie können auch einige Worte eintippen, dann ist das Scrolling besser sichtbar. Jedes Wort schließen Sie bitte mit <RETURN> ab.

Sie sehen, daß der Cursor zwischen der Spalte "Bezeichnung" und der Spalte "Wert" hin und her springt. Tippen Sie solange <RETURN> ein, bis der Cursor ungefähr in der Zeile mit der laufenden Nummer 5 steht.

Probieren Sie nun, was die Tasten <Pfeil nach oben> und <Pfeil nach unten> bewirken. Sie sehen, daß sich alle Zeilen entweder nach oben oder nach unten bewegen. Wenn Sie den Cursor ganz an das obere Ende der Datei bewegen, sehen Sie, daß überhalb der Eingabezeile mit der laufenden Nummer 1 vier Leerzeilen stehen. In diesem Augenblick hat die Variable 'Zeile1' den Wert 1. Sie arbeiten in der Zeile mit der laufenden Nummer 1, und die erste Zeile auf dem Bildschirm ist mit den ersten Elementen der drei Felder ('Nummer$', 'Bez$' und 'Zahl$') gefüllt. Stehen Sie mit dem Cursor in der Zeile Nummer 5, hat 'Zeile1' den Wert 5. Und so weiter. Wenn Sie dieses Prinzip verstanden haben, ist der Rest des Programms nicht mehr schwer.

Die nächsten beiden Programmzeilen begrenzen das Scrolling nach oben und nach unten: Vor die erste Zeile und hinter die letzte Zeile ('HoechstZeile') darf sich der Cursor nicht bewegen. Der Anwender wird mit einem BEEP darauf hingewiesen.

Das Programm druckt nun zuerst die Spaltenüberschriften ("Nummer", "Bezeichnung" und "Wert"), dann darunter die neun Datenzeilen. Dazu wird mal wieder ein neuer BASIC-Befehl verwendet: TAB kommt vom Wort "Tabulator". Vielleicht kennen Sie Tabulatoren von Schreibmaschinen oder sogar von einem Textverarbeitungsprogramm. Den Befehl TAB verwenden Sie, wenn Sie wollen, daß eine Bildschirmausgabe in einer bestimmten Spalte beginnt.

```
PRINT TAB(10);"Hallo!"
```

druckt das Wort "Hallo!" ab der 10. Bildschirmspalte. Mit dem TAB-Befehl können Sie sehr einfach Tabellen und Listen erzeugen.

Gegen Ende der 'Hauptschleife:' kommt noch eine Variable namens 'BezWert' ins Spiel. Sie erkennen darin wahrscheinlich schon die beiden Bestandteile "Bezeichnung" und "Wert". Die Variable 'BezWert' dient als Merker dafür, ob gerade eine Bezeichnung (in der Text-Spalte) oder ein Wert (in der Zahlen-Spalte) eingegeben wird.

Dem Unterprogramm 'Eingabe:' teilen wir außerdem in den beiden Variablen 'StartSpalte' und 'EndSpalte' mit, wo das erlaubte Eingabefeld beginnt und wo es endet. Der Wert 'xp' schließlich wird auch noch vom 'Eingabe:'-Programm verwendet. Er gibt die horizontale Position des Cursors an. Zu Beginn der Eingabe ist er identisch mit der Startspalte.

Nach allen Vorbereitungen rufen wir das 'Eingabe:'-Unterprogramm auf. Nach der Rückkehr wird 'in$' gelöscht. Das ist der Eingabestring, der fürs nächste Mal wieder leer sein soll.

Zuletzt springt die 'Hauptschleife:' an ihren eigenen Anfang. Solange keine Unterbrechung auftritt, läuft diese Schleife endlos.

Schauen wir uns jetzt das 'Eingabe:'-Unterprogramm näher an. Zuerst zu seiner grundsätzlichen Funktion: In diesem Programm haben wir die Routinen für die Tastatur-Eingabe selbst geschrieben. Normalerweise bietet AmigaBASIC ja Befehle wie INPUT und LINE INPUT. Aber während einer dieser Befehle eine Eingabe annimmt, kann AmigaBASIC nicht prüfen, welche Tasten gedrückt werden. Da wir zu jeder Zeit die volle Kontrolle über alle Tasten haben wollen, fragen wir Taste für Taste mit INKEY$ ab und setzen daraus bei Bedarf die Worte und Zahlen zusammen.

Das Programm beginnt mit einer Einschränkung für 'xp'. Diese Variable darf nicht kleiner werden als die 'StartSpalte'. Am Anfang ist das sowieso nicht möglich, schließlich haben wir noch kurz vor dem Aufruf 'xp' und 'StartSpalte' gleichgesetzt. Aber

wenn im Verlauf der Eingaben der Cursor nach links bewegt wird oder Zeichen gelöscht werden, kann es schon mal vorkommen, daß der Cursor das Eingabefeld an der linken Seite verlassen will. Dagegen hilft die erste Zeile.

Der Cursor bleibt in einer Bildschirmzeile fixiert, er kann nicht nach oben oder unten bewegt werden. Zeile 6 ist die Zeile, wo sich alles abspielt. Mit LOCATE 6,xp setzen wir den Cursor in die richtige Zeile auf die richtige horizontale Position. Der Cursor (wir müssen ihn in unserem Programm ja auch selbst erzeugen) soll ein oranges Kästchen sein, mit COLOR 0,3 wählen wir Orange als Hintergrundfarbe und Blau für den Vordergrund. Wenn wir nun ein einfaches Leerzeichen drucken, ist das Ergebnis ein oranges Kästchen. Danach wird die Farbe für alle anderen Textdarstellungen auf ihre Normaleinstellung zurückgesetzt.

Es folgt der zentrale Befehl der ganzen Eingaberoutine: i$=INKEY$. Die Variable 'i$' enthält immer das aktuelle, eingegebene Zeichen. Ist i$ ein Leersting, wurde keine Taste gedrückt. In diesem Fall springt das "Eingabe:"-Programm sofort wieder an seinen eigenen Anfang und wartet solange, bis eine Taste gedrückt wurde.

Übrigens: Falls Sie mal schneller tippen sollten, als es der Amiga verarbeiten kann, gehen trotzdem keine Zeichen verloren. AmigaBASIC hat einen 15 Zeichen großen Tastaturpuffer. Hier merkt sich der Amiga die Tastenanschläge, die nicht sofort bearbeitet werden konnten. Und INKEY$ liest grundsätzlich aus dem Tastaturpuffer. Wenn der Puffer voll ist, und Sie tippen weiter, werden Sie durch ein kurzes Aufblitzen auf dem Schirm gewarnt, daß ab jetzt Zeichen verloren gehen. Dasselbe Aufblitzen unterstützt durch ein Piepsen erleben Sie auch, wenn AmigaBASIC kein Window hat, dem es die Tastatureingaben zuordnen könnte.

Der Rest des 'Eingabe:'-Programms ist mit der Auswertung der eingegebenen Zeichen beschäftigt. Wenn Sie den Inhalt von INKEY$ (bzw. dessen Kopie in 'in$') prüfen wollen, brauchen Sie dazu den CHR$-Befehl. CHR$ macht ja aus einem ASCII-Code

das zugehörige Zeichen. Sie vergleichen Zeichen mit Zeichen, nämlich INKEY$ mit CHR$(x). Ist das Ergebnis eines solchen Vergleichs "wahr", kann Ihr Programm darauf reagieren.

CHR$(1) entspricht dem Zeichen <CTRL>-<A>. Dieses Zeichen entsteht, wenn Sie die <CTRL>-Taste gedrückt halten und dann die Taste <A> drücken. Wie viele Control-Zeichen (CTRL ist eine Abkürzung für "Control") erzeugt <CTRL>-<A> im BASIC-Window kein sichtbares Zeichen. Daß diese Zeichen aber dennoch etwas bewirken können, merken Sie, wenn Sie zum Beispiel im BASIC-Window <CTRL>-<G> drücken. Dieses Zeichen läßt den Piepser erklingen, den Sie schon von BEEP kennen. Sie kennen auch schon <CTRL>-<C>. Diese Tastenkombination bricht das aktuelle BASIC-Programm ab.

<CTRL>-<A> bekommt eine besondere Funktion innerhalb unseres Programms: A steht für Anfang, mit der Tastenkombination <CTRL>-<A> können Sie jederzeit an den Anfang Ihrer Datenliste springen, also in die erste Zeile. 'Zeile1' wird auf 1 gesetzt, das war's schon. Rücksprung mit RETURN, denn für die Anzeige der Zeilen ist die 'Hauptschleife:' zuständig. Genauso funktioniert <CTRL>-<E> bzw. CHR$(5). Damit können Sie ans Ende Ihrer Datei springen. Das Ende der Datei ist die höchste Zeile, wo bereits eine Eingabe erfolgte. Es darf sich dabei auch um eine Leereingabe handeln.

CHR$(12) entspricht <CTRL>-<L>. Mit dieser Tastenkombination können Sie die Zeile, in der der Cursor steht, aus der Datenliste löschen. Dafür gibt es ein eigenes Unterprogramm, es heißt 'Loeschen:'. Nach der Ausführung dieser Routine ebenfalls Rücksprung in die 'Hauptschleife:'. CHR$(14) schließlich, auf der Tastatur durch <CTRL>-<N> zu erzeugen, fügt eine neue Zeile an der aktuellen Cursorposition ein. Auch dafür gibt es ein eigenes Unterprogramm, nämlich 'Einfuegen:'.

Bei den vier Control-Funktionen in unserem Programm haben wir versucht, die Tasten so zu belegen, daß Sie auf die Funktion schließen können:

<CTRL>-<A> springt an den Anfang
<CTRL>-<E> ans Ende
<CTRL>-<L> Löscht eine Zeile
<CTRL>-<N> fügt eine Neue Zeile ein

Diese Hilfe für den Benutzer sollten Sie in Ihren Programmen auch anstreben. Woher kann ein unbedarfter Benutzer schließlich wissen, daß <CTRL>-<U> z.B. für "Hilfsmenü" stehen soll? Wo es möglich ist, sollten Sie anstelle von <CTRL>-Sequenzen sowieso lieber Pulldowns einsetzen, dort können Sie Klartext anzeigen, und die Bedienung mit der Maus macht eh mehr Spaß als über die Tastatur.

Die nächsten beiden Zeilen passen auf die Tasten <Cursor nach oben> und <Cursor nach unten> auf. CHR$(28) entspricht der <Cursor nach oben>-Taste. Wenn der Anwender diese Taste drückt, verläßt er ja das aktuelle Eingabefeld. Also soll die dort stehende Eingabe in die Liste übernommen werden, wofür 'Textuebernahme:' zuständig ist. 'xp', der Cursor-Positionszeiger, wird auf die Startspalte gesetzt, damit er im neuen Eingabefeld am Anfang steht. 'Zeile1' wird um 1 vermindert, und schon ist alles erledigt. Nach dem Rücksprung in die 'Hauptschleife:' wird um eine Zeile nach oben gescrollt.

CHR$(27) wird von <Cursor nach unten> erzeugt. Diese Zeile ist genauso aufgebaut wie die letzte, nur scrollen wir diesmal eine Zeile nach unten.

Die Variable 'TextPos' errechnet die Position des Cursors innerhalb des aktuellen Eingabetextes. Steht der Cursor beispielsweise in Spalte 14, und 'StartSpalte' ist 10, befinden wir uns auf der (14-10+1)-ten, sprich der fünften Position im eingegebenen Text. Sie können's gern nachzählen.

Im String 'Text$' steht der Text, den wir im gerade aktuellen Feld eingeben bzw. korrigieren. Je nach dem Inhalt von 'BezWert' wird dieser Text zu Beginn aus dem Feld 'Bez$' oder 'Zahl$' gelesen.

CHR$(30) wird durch <Cursor nach rechts> erzeugt. Steht der Cursor über einem bereits eingegebenen Text (was durch IF TextPos<=LEN(Text$) festgestellt werden kann) und der Anwender drückt die <Cursor nach rechts>-Taste, wird 'i$' jeweils das Zeichen aus 'Text$', über dem der Cursor steht.

CHR$(13) werden Sie auch in anderem Zusammenhang noch häufig brauchen, es ist der Code, den die <RETURN>-Taste erzeugt. CHR$(9) gehört zur <TAB>-Taste. Die beiden Tasten haben in unserem Programm dieselbe Funktion. Zunächst einmal zeigt <RETURN> bzw. <TAB> auf jeden Fall an, daß die Eingabe im aktuellen Feld abgeschlossen ist. Also wird 'Textuebernahme:' aufgerufen. 'BezWert' pendelt immer zwischen 1 und 0, um anzuzeigen, ob wir uns in der Spalte "Bezeichnung" oder in der Spalte "Wert" befinden. Die Formel (Variable)=1-(Variable) ändert den Wert einer Variablen bei jedem Durchlauf auf 1 oder auf 0.

Wenn 'BezWert' gerade den Wert 0 erhalten hat, wurde von der "Wert"-Spalte zurück in die Bezeichnungs-Spalte gewechselt. In diesem Fall führen wir auch noch einen Zeilenvorschub aus. 'xp' bekommt wieder den Spalten-Anfangswert. Gegebenenfalls wird noch eine neue 'HoechstZeile' festgelegt. 'HoechstZeile' gibt immer die höchste Zeilennummer an, in der bereits eine Eingabe erfolgte. Die Zeile, wohin der Cursor schon einmal vorgedrungen ist, ist auch die 'HoechstZeile'. Liegt der neue Wert von 'Zeile1' über der 'HoechstZeile', wird deren Wert angepaßt. Da nun der 'Eingabe:'-Teil seine Schuldigkeit getan hat, übergibt er durch RETURN den Rest der Arbeit an die 'Hauptschleife:'.

Der nächste Block prüft die Tasten <BACKSPACE> und <Cursor nach links> ab: Die beiden Tasten sind in unserem Programm auch gleichberechtigt. Mit ihnen löschen Sie die Zeichen links vom Cursor. Es können zwei Fälle auftreten: Befindet sich der Cursor innerhalb des Texts, der schon vorher in der Spalte war (IF TextPos<=LEN(Text$)...), soll beim Löschen neu eingegebener Zeichen der alte Text wieder zum Vorschein kommen. Gibt es keinen Text "unter" der Neueingabe, werden einfach Leerzeichen gedruckt. Der Ausdruck

```
RIGHT$(Text$,LEN(Text$)-TextPos+1)
```

liefert den Anteil von 'Text$', der sich hinter der aktuellen Cursorposition befindet.

Diese Arbeitsweise eines *Editors* ist Ihnen wahrscheinlich nicht so vertraut. Zuerst geben Sie die Texte oder Zahlen ein. Wenn Sie bei der Eingabe einen Fehler machen, müssen Sie zur direkten Korrektur solange mit <BACKSPACE> die hinteren Zeichen löschen, bis der Fehler gelöscht ist. Das kennen Sie von Amiga-BASIC aus dem BASIC-Window und von den Befehlen INPUT und LINE INPUT. Kommen Sie mit dem Cursor auf ein Feld, das bereits einen Text beinhaltet, können sie ihn über dem Wort nach links und nach rechts bewegen. Wenn Sie dabei ein neues Zeichen eingeben, wird das alte Zeichen durch das neue ersetzt. Bewegen Sie den Cursor aber über den neuen Text zurück, löschen Sie damit die neue Eingabe und die alte kommt darunter wieder zum Vorschein. Beim Drücken von <RETURN> wird der neue Text nur bis zur aktuellen Cursorposition übernommen, der Teil dahinter wird gelöscht.

Nach diesem Prinzip funktioniert zum Beispiel der Editor der Apple II-Computer, jener Computerveteranen, deren Ursprünge von Steve Jobs und Stephen Wozniak (den bisher wunderbarsten Wunderkindern der Computerbranche) in einer Garage entwickelt wurden. Die beiden haben damals fast alles allein gemacht, die ganze Hardware und die Software entwickelt. Dabei kam dann auch die etwas einfache, aber doch brauchbare Cursorsteuerung heraus, die sich natürlich nicht an den Bildschirmorientierten Editor-Versionen messen kann, wie sie die Amiga-Software teilweise bietet. Aber nach einer kurzen Eingewöhnung werden Sie sehen, daß man damit gar nicht so schlecht zurechtkommt. Ein noch komfortablerer Editor ist in BASIC jedenfalls nur sehr aufwendig zu erreichen.

Der Rest des Teils zum Löschen von Zeichen ist schnell erklärt: Die Cursorposition 'xp' wird um 1 vermindert, ist die 'StartSpalte' erreicht, piept's und der Cursor bewegt sich nicht weiter. Der Eingabestring 'in$' wird auf den Teil beschränkt, der links vom Cursor steht, also: in$=LEFT$(in$,(LEN(in$)-1)).

Ist alles erledigt, springt das Programm zurück zum Beginn des 'Eingabe:'-Teils, denn die Eingabe ist schließlich noch nicht beendet.

Als nächstes folgt ein kleiner Trick. Vielleicht haben Sie beim Ausprobieren des Programms schon bemerkt, daß Sie kein Anführungszeichen (") eintippen können. Alle anderen Zeichen sind erlaubt, aber anstelle des Anführungszeichens erscheint ein Apostroph (', dieses Zeichen wird auch "Hochkomma" genannt). Anführungszeichen im Text würden uns nämlich beim Abspeichern der Daten Schwierigkeiten machen, wir werden später erklären, warum das so ist.

Der letzte Teil der 'Eingabe:'-Schleife ist für die Eingabe ganz normaler Zeichen zuständig. Damit uns die Steuerzeichen wie CTRL-Codes oder Funktionstasten nicht ins Gehege kommen, haben wir die erlaubten Zeichen auf die ASCII-Codes von 32 bis 126 beschränkt. Sie können ja in der Tabelle im Anhang B nachschlagen, wenn Sie sich dafür interessieren, welche Zeichen das sind. Ist 'i$' eines dieser Zeichen und 'xp' nicht größer als 'EndSpalte' (sonst piepst's nämlich, und der Amiga verweigert die Annahme weiterer Zeichen in diesem Feld), dann wird das Zeichen auf den Bildschim gedruckt, und der Eingabestring 'in$' wird um das eingegebene Zeichen erweitert. Die Variable 'xp', die Cursor-Position, wird um 1 erhöht, dann erfolgt der Rücksprung zum Beginn der 'Eingabe:'-Schleife.

So weit, so gut. Uns fehlen noch die Unterprogramme, die der 'Eingabe:'-Schleife verschiedene Arbeiten abnehmen. Da gibt es zunächst einmal die 'Textuebernahme:'. Sie wird aufgerufen, wenn der Text der aktuellen Bildschirmeingabe in das zugehörige Datenfeld übernommen werden soll. Die ganze Routine wird vom Programm nur dann ausgeführt, wenn der Eingabestring 'in$' nicht leer ist - denn wo nichts ist, gibt es auch nichts zu übernehmen. Außerdem: Steht der Cursor auf der ersten Position eines Eingabefelds (was immer dann der Fall ist, wenn der Cursor gerade erst in das Feld gekommen ist) und wird dann <RETURN>, <TAB> oder eine der <Cursor nach

oben>/<Cursor nach unten>-Tasten gedrückt, soll am Feldinhalt nichts verändert werden. Das ermöglicht das Suchen und Bewegen innerhalb der Liste.

Hat 'in$' aber einen Inhalt, wird dieser Inhalt abhängig von 'BezWert' ins 'Bez$'-Feld oder ins 'Zahl$'-Feld übernommen. Danach löschen wir 'in$' für die nächste Eingabe. Schließlich bekommt noch die Variable 'AltDaten' den Wert 1. Mit ihr hat es folgende Bewandtnis: Eine Änderung der Daten kann nur stattgefunden haben, wenn das Unterprogramm 'Textuebernahme:' gelaufen ist. Will der Anwender das Programm verlassen und hat er die veränderten Daten noch nicht abgespeichert, weist ihn ein Warnungs-Window darauf hin. Zu Beginn des Programms hat 'AltDaten' den Wert 0. Solange es seinen Wert nicht ändert, wurden auch keine Daten verändert. Ist 'AltDaten' aber erst mal 1, so weiß das Programm, daß Änderungen durchgeführt wurden. Damit hat 'Textuebernahme:' schon alle Aufgaben erledigt, es springt zurück zur Zeile des Aufrufs.

Der nächste Teil heißt 'Loeschen:'. Er löscht eine Datenzeile. Um das zu erreichen, werden einfach alle Zeilen von der aktuellen Zeile bis zur letzten Zeile der Datei um eine Position nach vorn kopiert. Die Elemente der Felder 'Bez$' und 'Zahl$' bekommen die Inhalte der jeweils nächsthöheren Feldelemente. Die Zeile, in der der Cursor steht, verschwindet und wird durch die nächsthöhere Zeile ersetzt. Die 'HoechstZeile' vermindert sich um 1. Ist das obere Ende der Datei erreicht (IF HoechstZeile<1 THEN...), wird die 'HoechstZeile' auf die Anfangszeile gesetzt, also auf den Wert 1. Sind all diese Prozeduren erledigt, kann das Unterprogramm mit RETURN zurückspringen.

Die entgegengesetzte Funktion ist das 'Einfuegen:' von Datenzeilen, das vom nächsten Unterprogramm erledigt wird. Noch bevor irgend etwas anderes unternommen wird, prüft 'Einfuegen:', ob überhaupt noch Platz für eine weitere Zeile ist. Hat 'HoechstZeile' nämlich den Wert 50 erreicht, ist die Liste bereits gerammelt voll, und es können keine weiteren Zeilen eingefügt werden. Ein Piepser weist den Anwender darauf hin. Tritt dieses Problem nicht auf, beginnt das Unterprogramm seine Arbeit: Das 'Einfuegen:' funktioniert genau andersrum wie das 'Loe-

schen:'. Vom Ende der Datei (HoechstZeile+4 liefert die entsprechende Feldposition) werden bis zur aktuellen Zeile die Feldelemente um eine Position nach hinten verschoben. Durch Anhängen von STEP -1 erreichen Sie, daß eine FOR...NEXT-Schleife rückwärts zählt. Nach der Schleife wird die so freigewordene Position gelöscht, damit hier neue Daten eingegeben werden können. 'HoechstZeile' hat sich um 1 erhöht, die Variable wird entsprechend angepaßt. Nach diesem Programmschritt folgt dann auch schon wieder der Rücksprung.

Kommen wir jetzt zur eigentlichen Existenzberechtigung unseres Programms: Den Routinen zum Laden und Speichern der Daten auf Diskette. Denn bisher haben wir zwar allen möglichen Komfort programmiert, aber ansonsten hat sich noch nicht viel getan.

```
DatenSpeichern:
  MENU 1,0,0 : MENU 2,0,0
  MENU OFF
  GOSUB Eingabename
  WINDOW 1
  IF Nam$="" THEN EndeSpeichern
  OPEN Nam$ FOR OUTPUT AS 1
    PRINT #1,HoechstZeile+4
    FOR x=1 TO HoechstZeile+4
      WRITE #1,Bez$(x)
      WRITE #1,Zahl$(x)
    NEXT x
  CLOSE 1

EndeSpeichern:
  MENU 1,0,1 : MENU 2,0,1
  MENU ON
  AltDaten=0
RETURN

DatenLaden:
  IF AltDaten=1 THEN GOSUB Abfrage
  MENU 1,0,0 : MENU 2,0,0
  MENU OFF
```

```
  GOSUB Eingabename
  WINDOW 1
  IF Nam$="" THEN EndeLaden
  FOR x=1 TO 58
    Bez$(x)=""
    Zahl$(x)=""
  NEXT x
  OPEN Nam$ FOR INPUT AS 1
    INPUT #1,Anzahl
    HoechstZeile=Anzahl-4
    FOR x=1 TO Anzahl
      INPUT #1,Bez$(x)
      INPUT #1,Zahl$(x)
    NEXT x
    Zeile1=HoechstZeile
  CLOSE 1

EndeLaden:
  WINDOW 1
  COLOR 1,0
  CLS
  PRINT "Nummer";TAB(10);"Bezeichnung";TAB(45);"Wert"
  FOR x=Zeile1 TO Zeile1+8
    PRINT Nummer$(x);TAB(10);Bez$(x);TAB(45);Zahl$(x)
  NEXT x
  MENU 1,0,1 : MENU 2,0,1
  MENU ON
  AltDaten=0
RETURN

Eingabename:
  Altname$=Nam$
  WINDOW 2,"Bitte Dateinamen eingeben:",(50,80)-(580,88),0,-1
  CLS
  LINE INPUT Nam$
  IF Nam$= "=" OR Nam$="*" THEN Nam$=Altname$
  WINDOW CLOSE 2
RETURN
```

Wenn Sie im "Datei"-Pulldown die Option "speichern" wählen, führt das Programm das Unterprogramm 'DatenSpeichern:' aus. Durch das Event Trapping kann dieses Speichern jederzeit erfolgen. Bedenken Sie, daß die Änderungen, die Sie in der aktuellen Cursorzeile vornehmen, erst dann in die Datei übernommen werden, wenn Sie <RETURN> oder <TAB> drücken. Und selbstverständlich werden alle neuen Änderungen erst dann endgültig auf Diskette übernommen, wenn Sie die Datei abspeichern. Zuerst unterdrückt 'DatenSpeichern:' die Menüabfrage und schreibt die Pulldowns in Geisterschrift. Dann wird das Unterprogramm 'Eingabename:' aufgerufen, das den Dateinamen liefern wird. Ist der Dateiname 'Nam$' leer, wird nicht gespeichert, sondern direkt das Label 'EndeSpeichern:' angesprungen.

Zum Speichern öffnet das Unterprogramm die Datei mit dem Namen aus 'Nam$' zum Schreiben. Zuerst schreiben wir die Anzahl der Daten in die Datei. Die fragliche Zahl läßt sich durch 'HoechstZeile'+4 berechnen. So oft wird dann auch die folgende FOR...NEXT-Schleife ausgeführt, die die Elemente der Dateien 'Bez$' und 'Zahl$' in die Datei schreibt. Dabei tritt ein bisher unbekannter Befehl auf: PRINT kennen Sie ja, aber WRITE?

Im Prinzip bewirken die beiden dasselbe: Sie schreiben Daten (Zahlen oder Texte) auf den Bildschirm oder in eine Datei. Die Unterschiede liegen im Detail. Vergleichen wir die beiden Befehle im BASIC-Window. Geben Sie ein:

```
PRINT 1,2,3;4;5;6
```

und

```
WRITE 1,2,3;4;5;6
```

Beim PRINT-Befehl wirkt ein Komma als Trennzeichen, um die Zahlen spaltenweise zu drucken. Eine Spalte ist 15 Zeichen breit. WRITE gibt die Kommas einfach mit aus. Ein Strichpunkt bedeutet bei PRINT, Daten direkt nebeneinander zu drucken

(wobei positive Zahlen wegen des fehlenden Vorzeichens eine Leerstelle mitbekommen). Bei WRITE werden Strichpunkte einfach in Kommas umgesetzt. Doch weiter im Text. Apropos Text:

```
PRINT "Hallo, ";"wie geht's?"
```

und

```
WRITE "Hallo, ";"wie geht's?"
```

bewirken auch nicht gerade dasselbe. WRITE kleidet Strings grundsätzlich in Anführungszeichen ein, PRINT macht das nicht.

Warum ist das alles wichtig? Vielleicht haben Sie sich schon einmal Gedanken darüber gemacht, wie AmigaBASIC eigentlich die Zeichen in sequentiellen Dateien verwaltet. Schließlich stehen dort alle möglichen Daten hintereinander. Woran erkennt AmigaBASIC beim Lesen, wo ein Begriff aufhört und der nächste beginnt? Dazu gibt es verschiedene Trennzeichen. Ein kurzer Ausflug in die Bürotechnik bzw. in die Welt der guten alten mechanischen Schreibmaschine trägt hier zur Erklärung bei.

Der Wagenrücklaufcode (CHR$(13)) ist so ein Trennzeichen. Daher hat die <RETURN>-Taste nämlich ihren Namen: Bei den herkömlichen mechanischen Schreibmaschinen bewirkte <RETURN>, daß der Wagen an seine Ausgangsposition zurückkehrte (engl: return) und das Papier eine Zeile nach oben schob. Die amerikanischen Computer-Ingenieure haben diese Bezeichnung für das Zeichen übernommen, das eine Zeile beendet (eben RETURN-Code, im ASCII-Code ist es CHR$(13)). Das Zeichen CHR$(10) hat eine ähnliche Funktion. Es heißt LINE FEED (Zeilenvorschub). Der Unterschied ist, daß hier zwar eine Zeile weiter geschoben wird, aber kein Wagenrücklauf stattfindet. Genau genommen müßten also CARRIAGE RETURN (das ist die Langform von RETURN: Carriage = Wagen) und LINE FEED immer paarweise auftreten, um einen Zeilenvorschub mit Wagenrücklauf zu erreichen. Tatsächlich genügt aber CHR$(10), er beinhaltet CHR$(13). Damit sind wir auch schon wieder beim Programmieren. Der PRINT-Befehl hängt diesen Code hinter ei-

ner Ausgabe an, falls ein Ausduck nicht mit einem Komma oder einem Strichpunkt endet. Und zwar tut er das unabhängig davon, ob Sie auf den Bildschirm oder in eine Datei drucken. In einer sequentiellen Datei stehen Daten, die Sie mit den Befehlen

```
PRINT #1, "Hallo"
PRINT #1, 1,2,3
PRINT #1, 4;5;6
```

geschrieben haben, so:

```
Hallo<LF> 1<13*SPACE> 2<13*SPACE> 3 <LF> 4  5  6 <LF>
```

<LF> steht für LINE FEED, <SPACE> ist die amerikanische Bezeichnung für die Leertaste. Was lernen wir daraus? Erstens: Folgt einer PRINT-Anweisung kein Trennzeichen, wird LINE FEED automatisch gesetzt. Steht eine Komma zwischen zwei Daten, erzeugt AmigaBASIC die Anzahl an Leerzeichen, die nötig ist, um zur nächsten Tabulatorspalte zu kommen. Stichpunkte schreiben Daten direkt hintereinander, die Leerräume sind wegen fehlender Vorzeichen entstanden.

Die eigentlichen Probleme treten erst dann auf, wenn Sie die Werte aus sequentiellen Dateien mit INPUT# wieder auslesen wollen. Dann liest INPUT# nämlich grundsätzlich bis zum nächsten Trennzeichen. Und als Trennzeichen werden eben auch Kommas akzeptiert.

Machen Sie einmal einen kleinen Versuch mit uns. Wir werden im BASIC-Window im Direktmodus eine Datei erzeugen:

```
open "Testdatei" for output as 1
a$="Hallo, wie geht's?"
b$="Test"
print #1,a$
print #1,b$
close 1
```

Wir haben also zwei Strings in eine sequentielle Datei geschrieben. Jetzt wollen wir sie wieder auslesen:

```
open "Testdatei" for input as 1
input #1,a$
input #1,b$
?a$
?b$
```

Nanu? Was ist denn jetzt los? 'a$' hat den Inhalt "Hallo", und in 'b$' steht "wie geht's?". Das Komma im String wurde also als Trennzeichen interpretiert. Von "Test" haben wir noch gar nichts gesehen. Aber das läßt sich ändern.

```
input #1,c$
?c$
close 1
```

Jetzt erst haben wir alle Werte aus der Datei gelesen. Durch das Komma in einem der Strings ist alles durcheinandergekommen. So etwas würde natürlich in unserem Programm nicht gerade für Ordnung sorgen, weil bald kein Feldinhalt mehr stimmen würde.

Was tut BASIC, um Trennzeichen innerhalb von Strings zu neutralisieren? Es kleidet den String in Anführungszeichen ein. Dasselbe tut auch WRITE. Deshalb schreiben wir die Daten mit WRITE auf Diskette, dann können wir uns darauf verlassen, daß alle Strings später wieder korrekt gelesen werden. Das einzige, was in den Strings nicht vorkommen darf, sind Anführungszeichen. Die würden ja wieder alles durcheinanderbringen. Aus diesem Grund haben wir in der 'Eingabe:'-Routine die Anführungszeichen verboten und durch das Zeichen ' ersetzt.

Uff! Das waren jetzt lange Erklärungen für zwei Zeilen Programm. Aber Sie haben jetzt viel Grundsätzliches über sequentielle Dateien gelernt. Das wird für Sie später beim Programmieren eigener Datenverwaltungsprogramme sehr nützlich sein.

Der Programmteil 'DatenSpeichern:' bietet nicht mehr viel Aufregendes: Die Datei wird mit CLOSE geschlossen, 'EndeSpeichern:' reaktiviert die Menü-Überwachung und setzt 'AltDaten' auf 0. Letzteres deshalb, weil bis zur nächsten Eingabe an den Daten, die wir gespeichert haben, nichts mehr verändert wurde.

Was gespeichert wurde, soll ja später auch mal wieder eingelesen werden können. Dazu haben wir das Unterprogramm 'DatenLaden:'. Einlesen neuer Daten bedeutet gleichzeitig auch Löschen der alten. Um geänderte Daten davor zu schützen, fragen wir 'AltDaten' ab. Wurden Änderungen durchgeführt, weist der Programmteil 'Abfrage:' darauf hin und gibt dem Anwender Gelegenheit, die Daten doch noch zu speichern. Danach deaktivieren wir Menu Trapping, das ist ja mittlerweile nichts Ungewöhnliches mehr. Das Unterprogramm 'Eingabename:' liefert einen Dateinamen. Ist dieser Name leer, wird das Laden durch Ansprung von 'EndeLaden:' abgebrochen.

Die folgende FOR...NEXT-Schleife löscht die aktuellen Daten aus den Datenfeldern. Würden Sie nämlich eine Datei einlesen, die kürzer ist als die, die Sie gerade im Speicher haben, wäre ein Rest der alten Daten in der Liste. Da das nur in den wenigsten Fällen erwünscht sein dürfte, löschen wir vor dem Laden alle Daten.

Dann öffnen wir die angegebene Datei zum Lesen und lesen zunächst die Anzahl der Daten in der Datei ein. Da die vier leeren Feldelemente am Anfang der Datei mit abgespeichert wurden, bekommt 'HoechstZeile' den Wert 'Anzahl'-4. Alle Daten werden von einer FOR...NEXT-Schleife in die Felder 'Bez$' und 'Zahl$' eingelesen. Nach dem Laden soll der Cursor in der letzten Zeile der Datei stehen, damit bei fortgeschriebenen Listen (z.B. Kurs- oder Verkaufs-Entwicklungen) die neuen Daten gleich hinten angehängt werden können. Wir setzen dazu 'Zeile1' auf den Wert von 'HoechstZeile'.

'EndeLaden:' beinhaltet ein paar Zeilen, die schon einmal in der 'Hauptschleife:' vorkamen. Die aktuelle Liste wird hier auf den Bildschirm gebracht. Das ist deshalb nötig, weil die 'Hauptschleife:' erst wieder nach einem Tastendruck aktiviert würde.

Der Anwender soll aber die neuen Daten schon vorher auf dem Bildschirm sehen. Menu Trapping wird wieder zugelassen, und 'AltDaten' bekommt den Wert 0. Die gerade eingelesenen Daten wurden ja bisher noch nicht verändert und stehen in derselben Form auf Diskette. Danach erfolgt dann der Rücksprung aus dem Unterprogramm.

Das Unterprogramm 'Eingabename:' kennen Sie in fast identischer Form aus dem Malprogramm. Zur Erinnerung: Wenn Sie = oder * eingeben, wird der zuletzt verwendete Name weiterverwendet. Nach der Eingabe verschwindet das entstandene Window wieder vom Bildschirm.

Damit hätten wir auch die Programmteile zum Laden und Speichern von Daten eingegeben und besprochen. Jetzt fehlt nicht mehr viel, hauptsächlich das Unterprogramm, das den Anwender darauf hinweist, daß die aktuellen Daten noch nicht abgespeichert sind.

```
Abfrage:
  WINDOW 2,"Achtung!",(155,50)-(475,135),0,-1
  COLOR 0,1
  CLS
  LOCATE 2,3
  PRINT "Die aktuellen Daten wurden"
  PRINT "  noch nicht gespeichert."
  PRINT : PRINT "  Wollen Sie sie abspeichern?"
  LOCATE 9,13 : PRINT "Ja"
  LOCATE 9,25 : PRINT "Nein"
  LINE (95,57)-(148,74),0,b
  LINE (183,57)-(236,74),0,b
  BEEP
WarteMaus:
  Test=MOUSE(0)
  WHILE MOUSE(0)=0
  WEND
  x=MOUSE(1) : y=MOUSE(2)
  IF 95<x AND x<146 AND 57<y AND y<74 THEN
    PAINT (97,59),3,0
    GOSUB DatenSpeichern
```

```
      PAINT (97,59),1,0
      WINDOW CLOSE 2
      RETURN
    END IF
    IF 138<x AND x<236 AND 57<y AND y<74 THEN
      PAINT (185,59),3,0
      WINDOW CLOSE 2
      RETURN
    END IF
    GOTO WarteMaus

  DatenLoeschen:
    IF AltDaten=1 THEN GOSUB Abfrage
    RUN

  Ende:
    IF AltDaten=1 THEN GOSUB Abfrage
    COLOR 1,0
    MENU RESET
    CLS
  END
```

Die Arbeitsweise von 'Abfrage:' ist seit dem Malprogramm auch kein Rätsel mehr für uns: Ein Window wird erzeugt, in blauer Schrift auf weißem Grund erscheint der Text "Die aktuellen Daten wurden noch nicht gespeichert. Wollen Sie sie abspeichern?", und je ein Kästchen für Ja und Nein werden gezeichnet. Der ganze Auftritt wird akustisch von einem BEEP untermalt.

'WarteMaus:' wartet auf einen Mausklick und wertet ihn aus. Die MOUSE-Befehle können Sie, wie Sie sehen, auch ohne Event Trapping benutzen. Klickt der Anwender ins Ja-Feld, wird das Feld orange ausgemalt und 'DatenSpeichern' aufgerufen. Vorsicht: Wenn Sie beim Dateinamen eine leere Eingabe machen, sind Ihre Daten verloren, sie werden nicht abgespeichert. Nach der Rückkehr vom Speichern malen wir das Kästchen in seiner Ursprungsfarbe aus, schließen das Window und springen zurück.

Beim Klick ins Nein-Feld wird das Kästchen ebenfalls orange ausgemalt, dann das ganze Window geschlossen und ebenfalls zurückgesprungen. Erfolgte der Klick irgendwo anders im Window, rufen wir erneut 'WarteMaus:' auf.

Ganz zum Schluß noch zwei kleine Funktionen: 'DatenLoeschen:" dient zum Löschen des Speichers, alle Daten gehen verloren. Die schnellste und effektivste Methode, alle Variablen mit ihrer Grundbelegung zu laden, ist der erneute Start des Programms mit RUN. RUN innerhalb eines Programms startet das Programm von vorn, allerdings gehen dabei alle Variablen und Feldinhalte verloren. In unserem Fall ist das ausdrücklich erwünscht, aber das wird nicht immer der Fall sein. Normalerweise sollten Sie mit GOTO arbeiten, wenn Sie zum Programmanfang verzweigen wollen. Ach ja, fast hätten wir's vergessen: Vorher findet gegebenenfalls noch eine Sicherheitsabfrage statt.

Das ist bei 'Ende:' genauso. Über dieses Unterprogramm wird das Programm beendet. Dazu setzen wir die Hintergrundfarbe und die Textfarbe auf ihre Grundeinstellung zurück, reaktivieren die BASIC-Pulldowns und löschen den Bildschirm. Durch END kehren wir schließlich in den Direktmodus zurück.

Damit hätten Sie's eigentlich schon geschafft. Es fehlt nur noch der wichtigste Teil. Denn jetzt haben wir zwar Zahlen, aber noch keine Grafiken. Aber keine Sorge, Ihr Arbeitsaufwand beschränkt sich dabei auf ein Minimum. Bitte speichern Sie zuerst die gegenwärtige Version des Statistikdaten-Programms auf Diskette ab, wenn Sie es selbst eingegeben haben. Sicher ist sicher... Die Grafik-Routinen haben wir ja vor dem Eingeben des Programms im ASCII-Format abgespeichert. Wir hängen sie jetzt mit MERGE hinter unser Programm, und schon ist alles komplett. Wenn Sie unseren Schritten zum Beginn dieses Kapitels gefolgt sind, müßte sich die ASCII-Version Ihres Balken-/Tortengrafik-Utilities in der "Daten"-Schublade befinden. Schauen Sie aber sicherheitshalber mit FILES nochmal nach. Sie können dann auch gleich den richtigen Dateinamen herausfinden. Was nun zu tun ist, wissen Sie sicher noch. Geben Sie im BASIC-Window ein:

```
merge "BalkenTorten"
```

Nach einigen Sekunden befinden sich die Grafik-Routinen am Ende Ihres Programms. Speichern Sie bitte den so entstandenen Programmkomplex sofort wieder auf Diskette ab. Sie können dabei ruhig den Namen verwenden, den Sie schon für die grafiklose Version benutzt haben.
Das Ergebnis des ganzen Vorgangs finden Sie, wie schon erwähnt, auch unter dem Namen "BTDaten" in der "Daten"-Schublade der beigelegten Diskette.

Wenn Sie das Listing selbst eingegeben haben, sollten Sie das Programm auf Herz und Nieren nach Tippfehlern austesten. Klappt die Umsetzung in Grafiken, das Laden und Speichern usw.? Sie wissen ja: Bei Fehlermeldungen immer die Fehlerzeile mit unserem Listing vergleichen, bei Funktionsfehlern sollten Sie die zuständigen Programmteile untersuchen.

Was Sie zur Bedienung wissen müssen, haben Sie größtenteils schon in den Erklärungen erfahren. Nochmal eine kurze Zusammenfassung: Das Programm erzeugt ein Window, das ungefähr den halben Bildschirm einnimmt. Sie sehen einen orangen Cursor, der zu Beginn in der Spalte "Bezeichnung" in Zeile 1 steht. In dieser Spalte geben Sie die Bezeichnungen zu den Daten ein. Diese Texte werden später neben den Tortensegmenten bzw. unter den Balken gedruckt. Eine Eingabe können Sie mit <RETURN>, <TAB> oder <Cursor nach oben/unten> abschließen. In der Spalte "Wert" geben Sie den zugehörigen Wert (die Zahlen, prozentualen Anteile etc.) an. Wenn der Cursor auf dem ersten Zeichen einer Eingabe steht, wird die Eingabe nicht verändert. Sonst übernimmt das Programm den Text bis zur Cursorposition. Der Teil hinter dem Cursor wird gelöscht. Mit <BACKSPACE> oder <Cursor links> können Sie Zeichen löschen, ein evtl. darunterliegender Text kommt wieder zum Vorschein.

Auf dem Bildschirm wird ständig ein neun Zeilen großer Ausschnitt aus der Datenliste gezeigt, die ganze Liste umfaßt maximal 50 Elemente. Mit <CTRL>-<A> können Sie an den Listenanfang springen, mit <CTRL>-<E> ans augenblickliche Ende.

<CTRL>-<L> löscht eine Zeile, <CTRL>-<N> fügt eine neue Zeile ein. Um eine Datenliste zu speichern, wählen Sie "speichern" aus dem "Datei"-Pulldown. Die Option "laden" liest abgespeicherte Dateien ein. Mit "löschen" löschen Sie den Arbeitsspeicher des Programms, und mit "Programm beenden" bewirken Sie die Rückkehr in den BASIC-Direktmodus.

Um aufgrund der aktuellen Daten eine Grafik zu erstellen, wählen Sie die gewünschte Darstellungsform aus dem "Grafik"-Pulldown. Die Grafik baut sich auf, dann erscheint ein kleines Dialog-Window, das Sie auffordert, eine Taste zu drücken. Wenn Sie dieses Window stört, können Sie es in den Hintergrund klicken. Nach einem Tastendruck gelangen Sie zurück auf den Eingabe-Bildschirm.

Noch ein paar Tips zu Ihren Grafiken. Wenn es Ihnen darum geht, Daten optimal umzusetzen, sind diese Hinweise sicher recht nützlich: Bei Balkengrafiken sollten Sie die Bemerkungen sehr kurz halten, denn pro Balken ist ja nicht eben viel Platz vorhanden. Geben Sie vielleicht zusätzlich zu Ihren Daten eine Zeile ein, in der eine runde Zahl steht, die größer ist als alle anderen und so als Vergleichswert dienen kann. Ist Ihr höchster Wert z.B. 898, sollten Sie in einer Zeile den Wert 1000 angeben. Das macht die Skalierung (Einteilung) der Achse eindeutiger.

Wenn bei Tortengrafiken mehrere Kleinst-Werte vorkommen (bei Wahlhochrechnungen z.B. die sogenannten Splitterparteien), geben Sie deren Daten am besten zwischen den Großen ein. Wenn nämlich alle Kleinen am Ende kommen, kann es passieren, daß sich die Bemerkungstexte gegenseitig überschreiben. Auch bei Tortengrafiken sollten die Texte übrigens nicht zu lang werden. Auf der rechten Seite der Torte verschwinden sie hinter dem Bildschirmrand, auf der linken Seite ragen sie in die Grafik. Wenn Sie besondere Farben benötigen, schreiben Sie einfach unter den PALETTE-Befehl im Teil 'Vorbereitungen:' weitere PALETTE-Befehle.

Das war alles, was wir Ihnen noch sagen wollten. Jetzt wünschen wir viel Spaß mit Ihrem Statistikdaten-Programm. Hoffentlich machen Ihnen Ihre Daten genauso viel Spaß wie das Programm

und seine Grafiken. Allerdings kann an Ihren persönlichen Bilanzen und Erfolgsübersichten auch die schönste Grafik nicht viel ändern...

3.5 Amiga & Friends - Arbeit mit Peripheriegeräten

Im folgenden Kapitel geht es um die Geräte, die Sie an Ihren Amiga anschließen können oder schon angeschlossen haben. Aber selbst wenn Sie noch keinen Drucker besitzen und auch sonst nichts an die verschiedenen Schnittstellen Ihres Amiga gehängt haben, werden Sie einige interessante Möglichkeiten kennenlernen.

Übrigens, noch eine Erklärung zur Überschrift: *Peripheriegeräte* nennt man alle Zusatzgeräte, die man an einen Computer anschließen kann.

Das erste, was wir Ihnen vorstellen wollen, ist die RAM-Disk. Sie besitzen nämlich ein Diskettenlaufwerk mehr, als Sie bisher wahrscheinlich denken. Oder zumindest eine Art Ersatz für ein Diskettenlaufwerk. AmigaBASIC und AmigaDOS bieten gemeinsam die Möglichkeit, einen Teil des Amiga-Speichers wie ein Diskettenlaufwerk zu benutzen. Das bedeutet, Sie können dort Programme und Dateien abspeichern und von dort Programme und Daten einlesen. Die RAM-Disk kann mit Inhaltsverzeichnissen und Unterinhaltsverzeichnissen eingerichtet werden wie Disketten auch. Bei alledem ist zu keiner Zeit irgendein Stück Mechanik im Spiel. Alle Informationen stehen statt auf einer Diskettenoberfläche in den Speicherbausteinen. Sie wissen ja noch: RAM heißt "Random Access Memory", deutsch: Schreib-/ Lesespeicher.

Wie effektiv Sie die RAM-Disk einsetzen können, hängt natürlich auch davon ab, wieviel Speicherplatz Sie übrig haben. Denn der Speicher, den die RAM-Disk braucht, geht für AmigaBASIC und die anderen Progamme verloren. Die Speicherverwaltung der RAM-Disk ist allerdings dynamisch, das heißt, sie nimmt immer nur soviel Platz in Anspruch wie wirklich unbedingt nötig. Wenn Sie ein Programm von der RAM-Disk löschen, wird der Spei-

cher, der von dem Programm belegt war, sofort wieder freigegeben. Nach all der Theorie nun ein wenig Praxis. Geben Sie im BASIC-Window ein:

```
chdir "ram:"
```

Wenn Sie diesen Befehl zum ersten Mal verwenden, wird Ihr Amiga die Workbench-Diskette verlangen, denn von dort braucht er ein Hilfsprogramm, das die Arbeit mit der RAM-Disk unterstützt. Zumindest braucht er dieses Programm, wenn Sie auch auf der Workbenchoberfläche die RAM-Disk noch nicht eingesetzt haben.

Mit dem Befehl FILES können Sie sich wie gewohnt das Inhaltsverzeichnis anzeigen lassen. Es fällt auf, daß die RAM-Disk keinen Diskettennamen hat. Sie lesen nur "Directory of []". Möglicherweise steht im Inhaltsverzeichnis der RAM-Disk eine Datei namens "BasicClip". Wenn Sie Cut, Copy und Paste aus dem "Edit"-Pulldown benutzen, legt AmigaBASIC den Inhalt des Clipboards nämlich auf der RAM-Disk ab. Geben Sie im LIST-Window ein paar Zeilen ein. Es muß dabei kein neuer Software-Hit entstehen - ein paar einfache Zeilen Text genügen als Beispiel.

Dieses Programm oder was immer Sie sonst fabriziert haben, wollen wir jetzt abspeichern. Tippen Sie im BASIC-Window:

```
save "test"
```

Kaum haben Sie <RETURN> gedrückt, erscheint auch schon das OK auf dem Bildschirm. Die Zugriffs- und Übertragungszeiten auf der RAM-Disk sind sagenhaft schnell, weil die Dateien beim Lesen und Schreiben nur im Speicher hin- und herkopiert werden müssen. Schauen Sie sich bitte noch mal das Inhaltsverzeichnis der RAM-Disk an: Da gibt es jetzt die beiden Dateien "test" und "test.info". AmigaBASIC erzeugt ja beim Abspeichern automatisch zu jeder Datei eine Info-Datei. Die Workbench 1.2 stellt auch für die RAM-Disk ein Icon auf der Workbench-

Oberfläche dar. Dieses Icon erscheint, sobald die RAM-Disk zum erstenmal benutzt wird. Es ist dann allerdings bis zum Ausschalten des Amiga nicht mehr wegzubekommen.

Jetzt löschen Sie bitte den Speicher mit NEW und laden unsere Datei wieder von der RAM-Disk:

```
load "test"
```

Es ist schon beeindruckend, wie schnell die RAM-Disk arbeitet. Auch sehr lange BASIC-Programme sind in Sekundenbruchteilen geladen. Eine weitere Möglichkeit wäre, die Dateien eines Programms auf der RAM-Disk anzulegen und sie erst beim Beenden des Programms auf Diskette zu kopieren. Es gibt viele Möglichkeiten, die RAM-Disk sinnvoll einzusetzen. Allerdings sollten Sie bedenken: Im Gegensatz zu den Daten auf Diskette führen die Daten auf der RAM-Disk ein sehr gefährliches Leben. Einen Stromausfall oder einen Systemabsturz überleben sie nämlich nicht.

Der Gerätename "RAM:" ist aber nur einer von vielen. Kommen wir als nächstes zu den Druckern am Amiga. Falls Sie noch kein solches Peripheriegerät besitzen, sollten Sie die nächsten Seiten trotzdem nicht überblättern. Erstens kommt man manchmal schneller in den Besitz von Zusatz-Hardware, als man denkt, und zweitens schadet es nicht, die Befehle auf jeden Fall einmal gesehen zu haben.

Bevor Sie mit einem Drucker arbeiten können, sind einige Vorbereitungen nötig. Zuerst brauchen Sie ein Kabel, mit dem Sie Ihren Amiga und Ihren Drucker verbinden können. So ein Kabel sollte mittlerweile bei jedem Commodore-Fachhändler erhältlich sein. Es kostet zwischen DM 30,- und DM 70,-. Leute mit Hardware-Erfahrung können sich so ein Kabel auch selbst zusammenlöten. Laien seien aber ausdrücklich davor gewarnt: Schon bei einem kleinen Fehler könnten Sie Ihren Amiga und Ihren Drucker schwer beschädigen.

Als nächstes müssen Sie in Preferences, dem Einstellprogramm auf der Workbench, die Druckeranpassung einstellen, die zu Ihrem Drucker gehört. Für viele Drucker gibt es bereits fertige Anpassungsdateien auf der Workbench, an weiteren wird zur Zeit bei Commodore und bei einigen Druckerherstellern gearbeitet. Alles Nähere über die nötigen Arbeiten erfahren Sie im Amiga-Benutzerhandbuch in Kapitel 5.1.11. Erst wenn diese beiden Punkte erledigt sind, können Sie in AmigaBASIC mit Ihrem Drucker arbeiten. Vorher müssen Sie ihn natürlich noch einschalten, das ist der dritte vorbereitende Punkt.

Der einfachste BASIC-Befehl zur Ausgabe auf dem Drucker heißt LPRINT. Das L vor PRINT steht für "Line". Übersetzt könnte es "Zeile" meinen oder auch "Anschluß, Verbindung, Kabel". Die Probleme bei der Übersetzung von BASIC in die deutsche Sprache kennen Sie ja schon. Jedenfalls bewirkt LPRINT dasselbe wie PRINT, nur erfolgt die Ausgabe nicht auf dem Bildschirm, sondern auf dem Drucker:

```
lprint "Hallo, wie geht's?"
```

Vor dem ersten Drucken braucht Ihr Amiga die Workbench-Diskette. Von dort lädt er ein Hilfsprogramm zur Drucker-Ansteuerung, außerdem die von Ihnen eingestellte Druckeranpassung. Wenn alles geladen ist, druckt Ihr Drucker die Zeile "Hallo, wie geht's". Ab jetzt geht's schneller, nun weiß AmigaBASIC alles, was es über den Drucker wissen muß. Bei LPRINT dürfen Sie alle von PRINT bekannten Trennzeichen verwenden:

```
lprint "Hallo",5,"Amiga";"Test"
```

Endet eine Zeile mit einem Strichpunkt, wird sie nicht unmittelbar gedruckt, sondern im Drucker solange gespeichert, bis irgendwann ein Zeilenvorschubcode (CHR$(10)) folgt. Mit einem Drucker sind wir ja relativ nah an den Schreibmaschinen, von denen wir Ihnen erzählt haben, als es im letzten Kapitel um Steuercodes und Trennzeichen ging.

```
lprint chr$(10)
```

bewirkt einen Vorschub von zwei Zeilen: Der erste Zeilenvorschub wird durch das Steuerzeichen CHR$(10) erzeugt, und der zweite wird vom LPRINT-Befehl automatisch anhängt. Falls Ihr Drucker alle Ausgaben in derselben Zeile druckt, reagiert er nicht richtig auf den Zeilenvorschubcode. Überprüfen Sie in diesem Fall die Drucker-Einstellungen von Preferences und schlagen Sie auch mal im Druckerhandbuch nach. Manchmal muß die Stellung eines kleinen Schalters (ein sog. DIP-Switch) im Drucker geändert werden. Wenn Sie keine Lösung finden, hilft Ihnen sicher Ihr Fachhändler weiter.

Mit dem nächsten Befehl können Sie Programmlistings auf dem Drucker ausgeben. Das ist beim Entwickeln und Testen von Programmen besonders hilfreich. Laden Sie irgendein Programm, es sollte aber nicht zu lang sein. Eine unserer Vorspeisen ist gut geeignet. Geben Sie dann ein:

```
llist
```

Und schon druckt Ihr Drucker das Programm aus. AmigaBASIC ist während des Druckens blockiert. Bei längeren Druckausgaben, z.B. einem Listing unseres Malprogramms, kann das schon einige Zeit dauern. Sie können aber AmigaBASIC in den Hintergrund klicken und in der Zwischenzeit mit einem anderen Programm arbeiten. Gerade während des Druckens ist Multitasking sehr vorteilhaft. Das Ganze ist natürlich, wie immer, auch eine Frage des vorhandenen Speicherplatzes. Wer reich ist (und sei es nur an Bytes), hatte es schon immer etwas bequemer.

Möglicherweise besitzen Sie einen Drucker mit Pufferspeicher. Solche Drucker speichern die Druckausgaben in einem eingebauten RAM und vermitteln so dem Computer den Eindruck, das Drucken sei bereits beendet. AmigaBASIC ist wieder frei, und Sie können weiterarbeiten, obwohl der Drucker vielleicht noch eine Viertelstunde beschäftigt ist.

Im übrigen können Sie auch jeden Druckvorgang mit <CTRL>-<C> abbrechen.

Sie haben jetzt die Grundbefehle zur Druckerbenutzung kennengelernt. Doch damit ist das Repertoire von AmigaBASIC zum Thema Druck und Papier noch lange nicht erschöpft. Ähnlich wie "RAM:" gibt es auch für den Drucker Gerätenamen. Ja, sogar mehrere davon. Da wäre zunächst einmal "PRT:" (wie "Printer", Deutsch: Drucker). Eine andere Art, ein Programmlisting auszudrucken, ist der Befehl:

```
list ,"prt:"
```

Sie geben einfach hinter LIST an, wohin das Listing gedruckt werden soll. In der gleichen Weise können Sie ein Programm sogar in eine Datei auf Diskette listen lassen. Wir machen es zur Demonstration mal auf der RAM-Disk:

```
list ,"ram:Test"
```

Dieser Befehl erzeugt eine Datei namens "Test", in die das Programm geschrieben wird. Es steht dort im ASCII-Format. Der LIST-Befehl, gefolgt von einem Dateinamen, ist eine Alternative zum SAVE-Befehl mit ",a"-Option. Interessant ist in diesem Zusammenhang auch das Gerät "SCRN:" (=Screen, Deutsch: Bildschirm). Der Befehl

```
list ,"scrn:"
```

beschert Ihnen den seltenen Anblick eines echten Programmlistings im BASIC-Window.

Zurück zum Drucker: Neben "PRT:" gibt es für den Drucker auch die Bezeichnung "LPT1:". Die beiden Gerätenamen sind so gut wie gleichwertig. Daß es überhaupt zwei verschiedene Namen gibt, liegt daran, daß Microsoft möglichst hohe *Kompatibilität* zu seinen anderen BASIC-Versionen erreichen wollte. Beim IBM PC-BASIC von Microsoft ist "LPT1:" der "Line Printer 1", also der erste der angeschlossenen Drucker. Um die Umsetzung eines BASIC-Progamms vom IBM PC auf den Amiga zu erleichtern, versteht AmigaBASIC diese Druckerbezeichung und schickt die Ausgaben an seinen Standard-Drucker "PRT:".

Sie können die Gerätenamen auch benutzen, um eine Ausgabedatei zu öffnen und auf diesem Weg Daten an bestimmte Geräte zu schicken.

```
open "prt:" for output as 1
```

schickt alle Ausgaben, die in die Datei 1 geschrieben werden, auf den Drucker. Sollte sich AmigaBASIC hartnäckig mit der Fehlermeldung "File already open" gegen diesen Befehl wehren, steht schon eine Verbindung zum Gerät "PRT:". Es kann vorkommen, daß sich AmigaBASIC weigert, eine zweite Verbindung aufzubauen. Da wir zur Zeit aber sowieso nur experimentieren, schließen Sie, falls es Probleme gibt, am besten alle Dateien und Verbindungen mit dem NEW-Befehl. Probieren Sie dann:

```
print#1, "Hallo!"
```

Wundern Sie sich nicht, wenn auf dem Drucker noch nichts zu sehen ist. AmigaBASIC legt für Dateien grundsätzlich einen Pufferbereich an, um nicht jedes Zeichen einzeln übertragen zu müssen. Erst wenn dieser Puffer voll ist oder beim Befehl CLOSE wird der Inhalt übermittelt. Darin besteht auch der einzige kleine Unterschied zwischen "LPT1:" und "PRT:". "LPT1:" druckt sofort, während "PRT:" einen Puffer anlegt.

```
close 1
```

bringt den Drucker in jedem Fall zum Arbeiten.

Zu jedem beliebigen Gerät können Sie eine Datei öffnen und auf diese Weise Daten austauschen. Natürlich müssen Sie einen sinnvollen Datei-Modus wählen. Den Drucker FOR INPUT zu öffnen, findet AmigaBASIC zum Beispiel gar nicht witzig. Eine Fehlermeldung vom Typ "Bad file mode" oder "Device I/O error" ist die Folge. Die Buchstaben I/O sind übrigens sehr beliebt, wenn es um Datenaustausch geht. Sie stehen für Input/Output, zwei Begriffe, die Sie schon kennen. Und "Device" heißt übersetzt "Gerät".

Da wir in diesem Kapitel alle Gerätenamen kreuz und quer durcheinander verwenden, kann es vorkommen, daß sich AmigaBASIC schlichtweg weigert, eine Datei zu einem bestimmten Gerät zu öffnen. Notfalls müssen Sie die Workbench neu laden, dabei wird dann wirklich alles zurückgesetzt.

Zum Einlesen (FOR INPUT) können Sie aber zum Beispiel die Tastatur verwenden. Sie hat den Gerätenamen "KYBD:" (Keyboard, deutsch: Tastatur).

Das folgende Listing ist ein ganz simples Schreibprogramm: Die Zeichen, die Sie auf der Tastatur eingeben, erscheinen auf dem Bildschirm und auf dem Drucker.

```
OPEN "KYBD:" FOR INPUT AS 1
OPEN "SCRN:" FOR OUTPUT AS 2
OPEN "LPT1:" FOR OUTPUT AS 3
WHILE i$<>CHR$(138)
  i$=INPUT$(1,1)
  IF i$=CHR$(13) THEN i$=CHR$(10)
  PRINT #2,i$;
  IF i$=CHR$(8) THEN i$=CHR$(127)
  PRINT #3,i$;
WEND
CLOSE 1,2,3
```

Dieses Programm heißt auf unserer Diskette im Buch "Tipp-Printer".

Die Tastatur öffnen wir zum Lesen, den Bildschirm und den Drucker zum Schreiben. (Geht ja auch gar nicht anders.) CHR$(138) ist der Code, den die Funktionstaste <F10> erzeugt. Ein Druck auf diese Taste beendet das Programm. Den Befehl INPUT$ kennen Sie vielleicht noch vom Einlesen der Objekt-Daten. Wir werden ihn bald genauer besprechen. Vor der Ausgabe an den Drucker werden zwei besondere Zeichen umgesetzt: Ist das eingegebene Zeichen CHR$(13), also der Code der <RETURN>-Taste, wird es in CHR$(10) geändert, um beim Drucker einen Zeilenvorschub zu erreichen. Haben Sie CHR$(8) eingegeben (durch Drücken der <BACKSPACE>-Taste), machen

wir für den Drucker daraus CHR$(127). Das ist der Code, den die meisten Drucker zum Löschen eines Zeichens im Puffer verwenden. Jedes Zeichen wird ja von unserem Programm gleich nach der Eingabe an den Drucker übermittelt. Solange es noch nicht gedruckt ist, kann ein Fehler aber auf einfache Weise beseitigt werden: Einfach das letzte Zeichen im Puffer löschen.

Wenn das bei Ihrem Drucker nicht funktioniert, schauen Sie bitte im Drucker-Handbuch nach: In den meisten Handbüchern gibt es irgendwo eine Übersicht, welches Zeichen bzw. welcher Code was bewirkt. Solche Code-Tabellen sind grundsätzlich sehr interessant, denn hier finden Sie Informationen darüber, mit welchen Steuerzeichen Sie bei Ihrem Drucker die Schrift verändern oder Zeilenabstände einstellen können und vieles mehr.

Falls Sie zum Beispiel einen Epson-Drucker oder einen Epson-kompatiblen Drucker besitzen, gibt es jede Menge ESC-Sequenzen, mit denen der Drucker gesteuert werden kann. Was soll das schon wieder bedeuten? Was ist denn eine ESC-Sequenz? Ausgesprochen wird das Ganze "Escape-Sequenz". Und obwohl "Escape" eigentlich "verschwinden, fliehen" heißt, müssen Sie sich keine Sorgen machen, daß Ihr Drucker plötzlich sein Kabel an sich rafft und dann das Weite sucht. Der Effekt von ESC-Sequenzen ist ein ganz anderer.

Vielleicht haben Sie irgendwann einmal die Taste <ESC> auf der Amiga-Tastatur entdeckt? Mit dem Zeichen ESC (im ASCII-Code ist das CHR$(27)) hat es eine besondere Bewandtnis Als vor vielen Jahren in Amerika der ASCII-Code entwickelt wurde, gab es noch keine besonders vielseitigen Drucker. Man war schon froh, wenn so ein Gerät Groß- und Kleinbuchstaben erzeugen konnte. An Kursivschrift, Fettschrift, Breitschrift oder gar *Near Letter Quality* (alles Möglichkeiten moderner Matrix-Drucker) dachte noch niemand.

Folglich waren die Positionen der ASCII-Code-Tabelle, die nicht durch Buchstaben, Zahlen oder Satzzeichen besetzt waren (etwa 30 Codes), mehr als genug, um die wenigen Sondermöglichkeiten der Drucker zu aktivieren. So aktiviert CHR$(14) zum Beispiel bei vielen Druckern die Breitschrift. Heute reichen diese freien

Codes bei weitem nicht mehr aus. Besonders, da viele Codes (etwa CHR$(10) und CHR$(13)) sowieso für andere Aufgaben gebraucht werden.

Irgendwie mußte es also möglich sein, den Bereich an Steuer-Codes zu erweitern. Und dafür haben sich die Ingenieure in weiser Voraussicht das Zeichen Nummer 27 aufgehoben, den Escape-Code. Dieses Zeichen teilt dem Drucker mit, daß das nächste, nachfolgende Zeichen nicht gedruckt werden soll, sondern ein Steuerzeichen ist. Kommt CHR$(69) allein an, druckt der Drucker ein E. Kommt aber CHR$(27) und dann CHR$(69), also ESC E, weiß der Drucker: "Aha, das ist ein Steuerzeichen. Mal nachsehen, was ESC E zu bedeuten hat..."

Epson-kompatible Drucker schalten bei ESC E zum Beispiel auf Fettschrift um. Die Steuerzeichen-Tabelle der japanischen Firma Epson hat sich zu einem weit verbreiteten Standard entwicklelt. Deshalb gibt es schon viele Drucker, die diese einheitliche Steuerzeichen-Belegung verwenden.

Allerdings gibt es auch eine ganze Menge Drucker, die nicht Epson-kompatibel sind. Dafür kann es verschiedene Gründe geben: Vielleicht ist der Drucker-Hersteller der Meinung, er habe eine viel bessere Lösung zur Belegung der Steuercode-Tabelle gefunden. Oder der Drucker kann einfach viel mehr als die Drucker, die dem Epson-Standard folgen. (Besonders Laserdrucker sind meistens nicht Epson-Steuerzeichen-kompatibel.) Oder der Hersteller folgt einem anderen Standard, z.B. dem Standard, den die Firma IBM für ihre Drucker festgelegt hat.

Normalerweise hat das dann zur Folge, daß Programme, die mit Drucker-Steuerzeichen arbeiten, entweder nur mit bestimmten Druckern funktionieren oder sehr aufwendig auf möglichst viele Drucker-Standards angepaßt werden müssen. Stellen Sie sich vor, Sie kaufen irgendwann einen neuen Drucker, der zu Ihrem letzten Modell nicht kompatibel ist. In diesem Fall müßten Sie alle Ihre Programme ändern und auf die neuen Steuerzeichen anpassen.

Die Entwickler des Amiga haben eine viel bessere Lösung gefunden: Sie haben eine Standard-Druckercode-Tabelle festgelegt, an die sich alle Amiga-Programme halten sollen. Beim Ausdrucken werden diese Standard-Codes von der Drucker-anpassung, die Sie in Preferences festgelegt haben, so übersetzt, daß an den jeweils angeschlossenen Drucker genau die zu ihm passenden Steuerzeichen übermittelt werden. Bevor Sie zum ersten Mal drucken, wählen Sie also bitte den entsprechenden Drucker im Preferences-Druckermenü und speichern Sie Ihre Einstellung auf der Workbench-Diskette ab. Falls Ihr Drucker nicht in der Liste zu finden ist, brauchen Sie deshalb auch nicht gleich zu verzagen: Viele Drucker funktionieren auch mit einer Anpassung, die eigentlich für ein anderes Modell gedacht ist. Insbesondere die "Epson"-Anpassung gilt für viele Matrixdrucker. Am besten, Sie fragen gegebenenfalls Ihren Händler. Sollten Sie aber dennoch einen sogenannten "Exoten" als Drucker haben, sollten Sie sich mal an Commodore oder den Druckerhersteller wenden. Denn meistens sind die Hersteller beider Seiten ständig dabei, neue Druckeranpassungen für den Amiga zu entwickeln.

Jetzt aber wieder zurück zum Programmieren: Eine Preferences-Druckeranpassung funktioniert wie ein Dolmetscher; auf einer Seite kommt die Information in einer bestimmten Sprache hinein (bei uns die Standard-Steuerzeichen) und auf der anderen Seite kommt die Information übersetzt (in die Codes des jeweils gewählten Druckers) wieder heraus. So brauchen Sie sich beim Programmieren nie darum zu kümmern, welcher Drucker mit Ihrem Programm verwendet wird. Solange er in Preferences auftaucht, übernimmt Amiga die Übersetzungsarbeit.

Das alles nützt Ihnen natürlich nur, wenn Sie wissen, welche Druckersteuerzeichen Sie zur Verfügung haben. Deshalb haben wir eine Gesamtübersicht für Sie zusammengestellt. Schauen Sie sich dazu Tabelle 8 an. Wenn Ihnen einige Funktionen nichts sagen, lesen Sie mal im Handbuch Ihres Druckers nach, ob es die Funktion bei Ihrem Drucker überhaupt gibt und wie sie funktioniert.

Steuerzeichen:	Bedeutung:
CHR$(27)"c"	Initialisierung (Zurücksetzen) des Druckers
CHR$(27)"#1"	Abschaltung aller Sondermodi
CHR$(27)"D"	Zeilenvorschub, wie CHR$(10)
CHR$(27)"E"	Zeilenvorschub+Wagenrücklauf, wie CHR$(13)
CHR$(27)"M"	eine Zeile zurück
CHR$(27)"[0m"	normale Zeichendarstellung
CHR$(27)"[1m"	Fettdruck ein
CHR$(27)"[22m"	Fettdruck aus
CHR$(27)"[3m"	Kursiv ein
CHR$(27)"[23m"	Kursiv aus
CHR$(27)"[4m"	Unterstreichen ein
CHR$(27)"[24m"	Unterstreichen aus
CHR$(27);x;"m"	Vordergrundfarbe (x zwischen 30 und 39), Hintergrundfarbe (x zwischen 40 und 49)
CHR$(27)"[0w"	normale Schriftgröße
CHR$(27)"[2w"	Elite-Schrift ein
CHR$(27)"[1w"	Elite-Schrift aus
CHR$(27)"[4w"	Condensed (Schmalschrift) ein
CHR$(27)"[3w"	Condensed (Schmalschrift) aus
CHR$(27)"[6w"	Enlarged (Breitschrift) ein
CHR$(27)"[5w"	Enlarged (Breitschrift) aus
CHR$(27)"[2"CHR$(34)"z"	NLQ ein
CHR$(27)"[1"CHR$(34)"z"	NLQ aus
CHR$(27)"[4"CHR$(34)"z"	Doppeldruck ein
CHR$(27)"[3"CHR$(34)"z"	Doppeldruck aus
CHR$(27)"[6"CHR$(34)"z"	Schattenschrift ein
CHR$(27)"[5"CHR$(34)"z"	Schattenschrift aus

CHR$(27)"[2v"	Superscript (Hochstellen) ein
CHR$(27)"[1v"	Superscript (Hochstellen) aus
CHR$(27)"[4v"	Subscript (Tiefstellen) ein
CHR$(27)"[3v"	Subscript (Tiefstellen) aus
CHR$(27)"L"	Hochstellen (Halbschritt)
CHR$(27)"K"	Tiefstellen (Halbschritt)
CHR$(27)"[0v"	zurück zu Normalschrift
CHR$(27)"[2p"	Proportionalschrift ein
CHR$(27)"[1p"	Proportionalschrift aus
CHR$(27)"[0p"	Proportionalabstand löschen
CHR$(27)"[";x;"E"	Proportionalabstand = x
CHR$(27)"[5F"	Links ausrichten
CHR$(27)"[7F"	Rechts ausrichten
CHR$(27)"[6F"	Blocksatz
CHR$(27)"[0F"	Blocksatz aus
CHR$(27)"[3F"	Buchstabenbreite justieren
CHR$(27)"[1F"	Mittig zentrieren
CHR$(27)"[0z"	Zeilenabstand 1/8 Zoll
CHR$(27)"[1z"	Zeilenabstand 1/6 Zoll
CHR$(27)"[";x;"t"	Seitenlänge einstellen auf x Zeilen
CHR$(27)"[";x;"q"	Perforation überspringen um x Zeilen
CHR$(27)"[0q"	Perforation überspringen aus
CHR$(27)"(B"	amerikanischer Zeichensatz
CHR$(27)"(R"	französischer Zeichensatz
CHR$(27)"(K"	deutscher Zeichensatz
CHR$(27)"(A"	englischer Zeichensatz
CHR$(27)"(E"	dänischer Zeichensatz (Nr.1)
CHR$(27)"(H"	schwedischer Zeichensatz
CHR$(27)"(Y"	italienischer Zeichensatz
CHR$(27)"(Z"	spanischer Zeichensatz
CHR$(27)"(J"	japanischer Zeichensatz
CHR$(27)"(6"	norwegischer Zeichensatz
CHR$(27)"(C"	dänischer Zeichensatz (Nr.2)

CHR$(27)"#9"	linken Rand setzen
CHR$(27)"#0"	rechten Rand setzen
CHR$(27)"#8"	Seitenkopf setzen
CHR$(27)"#2"	Seitenfuß setzen
CHR$(27)"#3"	Ränder löschen
CHR$(27)"[";x;y;"r"	Seitenkopf x Zeilen von oben und Seitenfuß y Zeilen von unten
CHR$(27)"[";x;y;"s"	linken Rand (x) und rechten Rand (y) setzen
CHR$(27)"H"	Horizontalen Tabulator setzen
CHR$(27)"J"	Vertikalen Tabulator setzen
CHR$(27)"[0g"	Horizontalen Tabulator löschen
CHR$(27)"[3g"	Alle horizontalen Tabulatoren löschen
CHR$(27)"[1g"	Vertikalen Tabulator löschen
CHR$(27)"[4g"	Alle vertikalen Tabulatoren löschen
CHR$(27)"#4"	Alle Tabulatoren löschen
CHR$(27)"#5"	Standard-Tabulatoren setzen

Tabelle 8: Die Standard-Druckersteuerzeichen

Wenn Sie z.B. wollen, daß Ihr Drucker auf Fettschrift umschaltet, dann geben Sie ein:

```
lprint chr$(27);"[1m";"Hallo!"
```

Übrigens, bevor Sie eine Suchmannschaft losschicken: Bei der deutschen Tastaturbelegung erreichen Sie das Zeichen [, indem Sie gleichzeitig <ALT> und die <ü>-Taste drücken. Die Tastaturen der Amigas 500 und 2000 haben aber auch eigene Tasten für die eckigen Klammern. Sie finden sie im Ziffernblock.

Wenn Ihr Drucker in der Lage ist, fett zu drucken und Sie in Preferences die richtige Druckeranpassung gewählt haben, müßte das "Hallo" in Fettschrift ausgedruckt werden. Wenn Sie den Fettdruck wieder ausschalten wollen, geht das so:

```
lprint chr$(27);"[22m";"Hallo!"
```

Experimentieren Sie ruhig ein wenig mit den Steuerzeichen aus unserer Tabelle.

Natürlich können einige Drucker mehr als andere Drucker. Wenn Sie ein Steuerzeichen verwenden, das der angeschlossene Drucker nicht versteht, läßt es die Druckeranpassung einfach unter den Tisch fallen. Die gewählte Funktion wird also nicht ausgeführt - das nennt man dann nicht Befehlsverweigerung, sondern praktisches Denken.

Es könnte natürlich sein, daß Sie trotzdem vorhaben, Ihren Drucker mit seinen eigenen Steuercodes zu füttern. Dann macht uns die Steuerzeichen-Konvertierung (-Übersetzung) natürlich einen ziemlichen Strich durch die Rechnung... Es gibt aber doch eine Lösung: Sie können nämlich beliebige Zeichen ungefiltert direkt über die Schnittstelle übermitteln, an der Ihr Drucker angeschlossen ist. Falls Ihnen das Wort "Schnittstelle" nichts sagt: Gemeint sind alle Stecker und Buchsen am Amiga, an denen Zusatzgeräte angeschlossen werden können. Der Gerätename für die Parallel-Schnittstelle heißt "PAR:", der für die serielle Schnittstelle heißt "SER:".

Zu den beiden Schnittstellen nur soviel: Ihr Amiga hat mehrere Anschlüsse zur Datenübertragung. Drucker sind meistens an der Parallel-Schnittstelle angeschlossen, können aber vereinzelt auch an der seriellen Schnittstelle hängen. Eine parallele Schnittstelle besitzt mehrere Datenleitungen, die Bits eines Zeichens werden gleichzeitig (parallel) übertragen. Eine serielle Schnittstelle hat nur eine Datenleitung, die Bits werden hintereinander (seriell) übertragen. Das ist der ganze Unterschied. Nehmen wir an, Sie haben Ihren Drucker am parallelen Interface angeschlossen. (Computerprofis nennen eine "Schnittstelle" gern "Interface".) Um beliebige Steuerzeichen übertragen zu können, öffnen Sie einfach eine Datei zum Gerät "PAR:"

```
open "par:" for output as 1
```

Wenn Sie unsere vorherigen Beispiele mitgemacht haben, wird sich AmigaBASIC hartnäckig weigern, diese Datei zu öffnen ("File already open"). Denn über die parallele Schnittstelle laufen

ja schon die Drucker "PRT:" und "LPT1:". AmigaBASIC stellt sich nun auf den Standpunkt, wenn dort eh' schon ein Drucker angeschlossen ist, kann kein zweites Gerät an derselben Schnittstelle hängen. Also ist das Gerät "PAR:" nicht verfügbar. Ende der Debatte!

Bei soviel Hartnäckigkeit hilft nur, die Workbench neu zu laden. (Das geht mit der Tastenkombination <CTRL>-<geschlossene Amiga-Taste>-<offene Amiga-Taste>.) Bei der Arbeit in AmigaBASIC müssen Sie sich zu Beginn entscheiden, ob Sie mit der gefilterten oder der ungefilterten Drucker-Anpassung arbeiten wollen. Wenn die Workbench geladen ist, legen Sie bitte Ihre "BASICDisk" ein und klicken Sie AmigaBASIC an. Wiederholen Sie dann die Eingabe von oben. Danach können Sie wie gewohnt Ihre Daten zum Drucker schicken:

```
print #1,chr$(27);chr$(69);"Fettschrift!"

close 1
```

Wenn Sie jetzt keine Fettschrift auf dem Druckerpapier sehen, versteht Ihr Drucker das Steuerzeichen nicht.

Ein Gerätename fehlt uns noch, dann haben wir alle komplett. (Nur zur Erinnerung, bisher kennen wir: "RAM:", "PRT:", "LPT1:", "PAR:" ,"SER:", "SCRN:" und "KYBD:".) Dieser letzte Name heißt "COM1:" Er wird benutzt, um eine besondere Schnittstellennorm (für Fachleute und solche, die es werden wollen: eine RS-232-Schnittstelle) am seriellen Interface zu aktivieren.

Wir nehmen an, daß nur die wenigsten unter Ihnen diese Schnittstelle zum gegenwärtigen Zeitpunkt brauchen, und dürfen das interessierte Publikum in den Anhang B verweisen. (Beim OPEN-Befehl finden Sie Näheres.)

Damit sind wir aber noch nicht ganz fertig. Eine Art von Peripheriegerät (zumindest im weiteren Sinne) haben wir noch nicht erwähnt: Die Joysticks. Solche Steuerknüppel findet man ja eher an Telespielen oder Home-Computern. Aber auch am Amiga

kann man sie anschließen und verwenden. Amiga-Spiele funktionieren sogar oft ausschließlich mit Joysticks. Da ist es schon fast selbstverständlich, daß auch AmigaBASIC die Steuerknüppel abfragen kann. Falls Sie einen besitzen, holen Sie ihn doch mal aus der Versenkung.

So ein Gerät besteht im wesentlichen aus einem Knüppel, den man in vier Richtungen bewegen kann und einem oder mehreren Feuerknöpfen. Schon diese Ausstattung läßt vermuten, daß Joysticks eher zum Fliegen von Raumschiffen als zur Bedienung von Wirtschafts-Programmen gedacht sind.

Stellt sich die Frage, wie Sie den Joystick mit Ihrem Amiga verbinden können. An der rechten Seite des Geräts befinden sich zwei Anschlüsse. Im Anschluß 1 steckt die Maus. Wir wollen sie auch erstmal dort steckenlassen, was wäre der Amiga schließlich ohne seine Maus? Ihren Joystick stecken Sie bitte in den zweiten Anschluß.

Wenn Sie ihn nun bewegen oder den Feuerknopf drücken, passiert zunächst noch gar nichts. AmigaBASIC nimmt überhaupt keine Notiz von dem neuen Eingabegerät. Das kann sich aber schnell ändern. Geben Sie bitte das folgende Programm ein:

```
OPEN "ExtrasD:BasicDemos/Ball" FOR INPUT AS 1
  OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
x=320 : y=100
OBJECT.ON 1

WHILE 1
  OBJECT.X 1,x
  OBJECT.Y 1,y
  x=x+STICK(2)
  y=y+STICK(3)
WEND
```

Speichern Sie dieses Programm bitte unbedingt erst auf Diskette ab und probieren Sie es dann aus. Sie finden eine Version auch unter dem Namen "JoystickDemo" in der "Daten"-Schublade unserer Diskette im Buch.

Wir wollen mit dem Joystick ein Grafikobjekt über den Bildschirm steuern. Dazu haben wir uns entschlossen, den "Ball" aus der "BasicDemos"-Schublade der Extras-Diskette zu laden. Wenn Sie Ihre eigenen Bobs oder Sprites vom Videotitel-Programm lieber haben, können Sie die natürlich auch verwenden. Wie das geht, wissen Sie ja.

Die Variablen 'x' und 'y' geben die Position des Objekts an. Sie werden zu Beginn des Programms ungefähr auf die Bildschirmmitte festgelegt. Dann machen wir mit OBJECT.ON das Grafikobjekt sichtbar. Die endlose WHILE...WEND-Schleife ist für die Bewegung zuständig. Und hier finden wir auch den neuen, ausschlaggebenden Befehl: STICK(x). Mit ihm ist es möglich, den Joystick auszulesen. Wie funktioniert das?

Am Amiga können zwei Joysticks angeschlossen sein. Einer im Anschluß 1 und ein anderer im Anschluß 2. Wir sprechen von Joystick 1 und Joystick 2. Jeder der Joysticks kann in einer X- und in einer Y-Richtng bewegt werden. Je nach der Zahl, die Sie in Klammern hinter STICK angeben, wird eine Bewegungsrichtung eines Joysticks überprüft:

STICK(0)	untersucht die X-Bewegung von Joystick 1
STICK(1)	untersucht die Y-Bewegung von Joystick 1
STICK(2)	untersucht die X-Bewegung von Joystick 2
STICK(3)	untersucht die Y-Bewegung von Joystick 2

STICK(x) ist eine BASIC-Funktion und kann drei mögliche Ergebnisse liefern.

Ergebnis	Bedeutung
0	keine Bewegung in der untersuchten Richtung
1	Bewegung nach oben bzw. nach rechts
-1	Bewegung nach unten bzw. nach links

Bewegen wir also den Joystick 2 nach oben, ist das Ergebnis von STICK(3) gleich 1. In unserem Programm addieren wir die Funktionsergebnisse einfach zur X- und zur Y-Variablen, dann ändern sie sich entsprechend der Joystick-Bewegung. Mit OBJECT.X und OBJECT.Y erscheint das Objekt an seiner neuen Position. Eine diagonale Bewegung entsteht, wenn eine X- und eine Y-Bewegung gleichzeitig stattfinden. Das kennen Sie schon in ähnlicher Form von den OBJECT-Befehlen.

Probieren Sie einfach unser Beispielprogramm aus. Dann merken Sie, wie die Bewegungen des Steuerknüppels die Bewegung des Grafikobjekts beeinflussen. Dann gibt es da noch den Feuerknopf. Ergänzen Sie Ihr Progamm, indem Sie innerhalb der WHILE...WEND-Schleife eine weitere Zeile einfügen:

```
IF STRIG(3) =-1 THEN BEEP
```

Wenn Sie den Feuerknopf drücken, gibt der Amiga einen Piepser von sich. Der Befehl STRIG(x) fragt die Feuerknöpfe der angeschlossenen Joysticks ab. STRIG ist ein Kürzel aus "Stick Trigger" ("Trigger" heißt Abzug, Feuerknopf). Auch bei STRIG(x) müssen Sie angeben, was Sie überprüfen wollen. STRIG(1) ergibt das Ergebnis -1, wenn der Feuerknopf von Joystick 1 gedrückt ist, für STRIG(3) gilt dasselbe, nur beim Joystick 2. Bleiben noch STRIG(0) und STRIG(2) übrig. Sie funktionieren ähnlich wie MOUSE(0) bei der Mausabfrage: Die Funktionen liefern dann das Ergebnis -1, wenn der jeweilige Feuerknopf seit der letzten Abfrage gedrückt wurde. So kann ein Spielprogramm auch feststellen, ob der Knopf betätigt wurde, während das Programm mit irgendetwas anderem beschäftigt war.

Wenn Sie Lust haben, können Sie jetzt mit diesen Funktionen einen neuen Spielhallen-Renner schreiben. Einen wichtigen Hinweis haben wir aber noch: Alles, was wir Ihnen vorgeführt haben, können Sie auch mit dem Joystick 1 machen. Aber Achtung: Wenn sich auch nur ein einziger Joystick-Befehl auf den Anschluß 1 bezieht, erwartet AmigaBASIC dort auch wirklich einen Joystick und erkennt die Maus nicht mehr, die normalerweise dort angeschlossen ist. Der Mauscursor bleibt an seiner Position stehen und kann nur noch über die Tastatur bewegt werden. (Falls Sie diese Möglichkeit noch nicht kennen: Eine der beiden <Amiga>-Tasten in Verbindung mit den Cursortasten ermöglicht die Steuerung des Mauscursors über die Tastatur.)

Nur damit Sie sehen, wie traurig so eine mauslose Zeit ist: Speichern Sie bitte alles ab und geben Sie dann den Befehl ein:

```
? stick(0)
```

Ab jetzt tut sich nichts mehr mit der Maus. Diese unfreundliche Eigenschaft von AmigaBASIC können Sie nur durch einen NEW-Befehl abschalten. Doch manchmal hilft auch das nicht mehr. Dann bleibt Ihnen nur noch übrig, die Workbench neu zu starten. Wenn man bedenkt, wie wichtig die Maus für den Amiga ist, muß man sich über die softwaremäßige Rücksichtslosigkeit, die Microsoft hier an den Tag legt, schon schwer wundern...

3.6 Was man Schwarz auf Weiß besitzt... - Eine Druckroutine fürs Statistikprogramm

Ihre neu erworbenen Kenntnisse über die Arbeit mit dem Drucker können Sie gleich für eine kleine Erweiterung des Statistik-Programms einsetzen. Wir wollen eine Routine schreiben, die die gesamte Statistik-Datei ausdruckt. Sie können dann z.B. alte Kurse oder Daten in schriftlicher Form aufheben. Oder die Daten in dieser Form einem Geschäftspartner vorlegen, wenn Sie keine Möglichkeit haben, die Grafik zu präsentieren. Und nachdem Sie ja zu solchen Anläßen schlecht den Amiga mitschleppen können, ist es am besten, einen Ausdruck zu machen.

Laden Sie bitte Ihr Statistik-Daten-Verwaltungsprogramm. Die Arbeiten, die zur Erweiterung nötig sind, sind eigentlich minimal. Das Ausdrucken wollen wir durch eine weitere Pulldown-Option realisieren, deshalb muß der Teil 'Vorbereitungen:' etwas verändert werden.

In der Version auf der beigelegten Diskette wurden die folgenden Änderungen schon ausgeführt.

```
MENU 1,0,1,"Datei"
MENU 1,1,1,"laden"
MENU 1,2,1,"speichern"
MENU 1,3,1,"drucken"
MENU 1,4,1,"löschen"
MENU 1,5,1,"Programm beenden"
...
```

Ebenso der Teil 'Menukontrolle:':

```
IF Men=1 THEN
  IF Menpunkt=1 THEN GOSUB DatenLaden
  IF Menpunkt=2 THEN GOSUB DatenSpeichern
  IF Menpunkt=3 THEN GOSUB DatenDrucken
  IF Menpunkt=4 THEN GOSUB DatenLoeschen
  IF Menpunkt=5 THEN Ende
```

Jetzt fehlt nur noch das zuständige Unterprogramm. Geben Sie es am besten unter dem Programmteil 'Eingabename:' ein:

```
DatenDrucken:
  MENU 1,0,0 : MENU 2,0,0
  MENU OFF
  OPEN "PRT:" FOR OUTPUT AS 1
    PRINT #1,"Datei:";Nam$;CHR$(10)
    PRINT #1,"Nummer";TAB(10);"Bezeichnung";TAB(45);"Wert"
    FOR x=4 TO HoechstZeile+4
      PRINT #1,Nummer$(x);TAB(10);Bez$(x);TAB(45);Zahl$(x)
    NEXT x
```

```
    CLOSE 1
    MENU 1,0,1 : MENU 2,0,1
    MENU ON
RETURN
```

Eigentlich dürfte dieses Programm für Sie keine Schwierigkeiten mehr beinhalten. Wir unterdrücken Menu Trapping, öffnen eine Druckerdatei, drucken zuerst den Namen der geladenen Datei (in 'Nam$' steht der zuletzt verwendete Dateiname), drucken dann die Kopfzeile und die einzelnen Datenzeilen. Danach schließen wir die Druckdatei, reaktivieren die Menü-Auswahl und kehren zurück zu der Zeile, die gerade bearbeitet wurde, als wir die Option "drucken" auswählten.

Wenn Sie Lust haben und über einen geeigneten Drucker verfügen, können Sie die Liste mit verschiedenen Schriftarten etc. nach Ihrem persönlichen Geschmack gestalten. Immer wenn wir Programme wie jetzt erweitern und verbessern, können Sie übrigens die Vorteile des AmigaBASIC besonders deutlich merken. Dank des modulartigen Aufbaues der Programme ist es wirklich ein leichtes, nachträglich neue Dinge zu programmieren und in das Programm einzufügen.

Sollte in Ihnen jetzt, nach all dem Druck, der starke Wunsch erwachen, auch die Grafiken auf Papier auszudrucken, dürfen wir Sie nochmals auf das "GraphicDump"-Programm in der "System"-Schublade der Workbench verweisen. In den nächsten Kapiteln werden wir Ihnen noch einige andere Möglichkeiten vorstellen, wie Sie Bilder und Grafiken der Nachwelt erhalten können.

4. Ein Bild sagt mehr als tausend Daten - Laden und Speichern von Grafiken

Was in unserem Malprogramm noch fehlt, wissen Sie sicher selbst: Es macht schließlich auf Dauer keinen großen Spaß, die wunderschönsten Bilder zu entwerfen, wenn man genau weiß, daß sie das nächste Ausschalten des Computers nicht überleben werden. Wir zeigen Ihnen, wie Sie Grafikdaten auf Diskette speichern können, um sie später jederzeit wieder einzulesen.

4.1 Aus eins mach zwei mach drei mach vier - die Befehle GET und PUT

Zu Beginn wollen wir zwei neue BASIC-Befehle vorstellen: GET und PUT. Was man mit Ihnen anfangen kann, zeigt am besten ein kurzes Beispielprogramm. Auf der beiliegenden Diskette hat es den Namen "GetPutDemo".

```
DIM Feld%(563)

CIRCLE (50,20),40
CIRCLE (35,12),5
CIRCLE (65,12),5
CIRCLE (50,20),8
CIRCLE (50,18),30,,4,6
PAINT (50,20),3,1

GET (0,0)-(99,39),Feld%

CLS
FOR x=0 TO 5
  FOR y=0 TO 4
    PUT (x*100,y*40),Feld%
  NEXT y
NEXT x
```

Da erscheint plötzlich eine ganze Armee von kleinen Gesichtern auf dem Bildschirm, obwohl wir nur ein einziges mit CIRCLE-Befehlen gemalt haben. Das Geheimnis muß hinter den beiden neuen Befehlen stecken. Mit dem Befehl GET können Sie einen Ausschnitt aus einer Grafik in einem Datenfeld abspeichern. In unserem Beispiel benutzen wir das Datenfeld 'Feld%'. Der Befehl PUT dient dazu, den gespeicherten Bildausschnitt an einer beliebigen Stelle des Bildschirms wieder darzustellen. GET liest einfach die Bits aus den verschiedenen Bitebenen, setzt sie zu Bytes zusammen und speichert die Bytes im angegebenen Datenfeld. PUT kopiert die Bytes an der angegebenen Stelle in die Bitebenen zurück. Das ist alles.

Zuerst müssen Sie ein Datenfeld dimensionieren. Mit einer Formel können Sie ausrechnen, wieviele Elemente für einen bestimmten Bildausschnitt benötigt werden. Diese Formel stellen wir Ihnen gleich vor. Beim GET-Befehl geben Sie in der gewohnten Schreibweise die Koordinaten der linken oberen Ecke und der rechten unteren Ecke des gewünschten Bildausschnitts an. Der Bildausschnitt muß rechteckig sein. Dahinter schreiben Sie noch den Namen des Datenfelds, in dem die Daten abgelegt werden sollen:

GET (xAnfang,yAnfang)-(xEnde,yEnde), Datenfeld

Um die gespeicherte Grafik an einer beliebigen Stelle des Bildschirms wieder sichtbar zu machen, braucht der PUT-Befehl nur die Koordinaten der linken oberen Ecke und den Datenfeldnamen:

PUT (xZielpunkt,yZielpunkt), Datenfeld

Schauen wir uns jetzt die versprochene Formel an, mit der Sie den Speicherbedarf eines Grafikausschnitts errechnen können. Sie lautet:

6 + Bitebenen * Höhe * 2 * INT((Breite + 16) / 16)

Die Werte für Höhe und Breite geben Sie in Pixels an. Der hintere Teil der Formel errechnet daraus die Anzahl an Bytes, die für einen Grafikausschnitt der angebenen Ausmaße benötigt werden. Diese Zahl muß mit der Anzahl der Bitebenen multipliziert werden, denn wie Sie wissen, gehören zu jedem farbigen Punkt mehrere Bits, die in verschiedenen Bitebenen liegen. Zuletzt kommen nochmal 6 Bytes dazu, die besondere Steuerinformationen enthalten werden. Als Ergebnis erhalten Sie den Speicherbedarf in Bytes. Diese Zahl ist noch nicht identisch mit der Anzahl an Feldelementen, denn ein Integerfeld wie unser 'Feld%' beinhaltet pro Feldelement zwei Bytes. (Sie wissen ja noch: Eine Integerzahl kann einen Wert zwischen -32768 und 32767 haben. Zur Speicherung braucht der Amiga 16 Bits, also 2 Bytes.)

Wie sieht die Berechnung für unser Beispiel aus? Die Höhe des Ausschnitts ist 40 Pixels, die Breite 100 Pixels. Berechnen wir alles von hinten her: (INT(100+16)/16) ergibt INT(116/16) also 7. Diese Zahl wird mit 2 multipliziert, Ergebnis: 14. Eine Bildzeile, die 100 Pixels breit ist, benötigt zur Speicherung 14 Bytes. 14 multipliziert mit der Höhe (40) ergibt 560. Wenn wir bloß eine Bitebene verwenden würden, bräuchte unsere kleine Grafik also 560 Bytes.

Da der Workbench-Screen standardmäßig mit 2 Bitebenen arbeitet, müssen wir die Zahl 560 verdoppeln und erhalten 1120. Dazu kommen noch die zusätzlichen 6 Bytes, der Gesamtspeicherbedarf beträgt also 1126 Bytes. Wenn unser 'Feld%' 1126 Bytes aufnehmen soll, müssen wir es auf 1126/2, also 563 Elemente dimensionieren. Sie sehen, der Speicherbedarf einer Grafik ist recht beachtlich. Denn so groß ist sie ja nun wirklich nicht. Trotzdem benötigt sie ein Vielfaches der Daten, mit denen zum Beispiel unser Statistikprogramm sein Dasein bestreitet.

Übrigens: Wenn Sie sich fragen, wozu eigentlich die 6 überzähligen Bytes gebraucht werden, schauen Sie sich einmal die erste drei Elemente von 'Feld%' an:

```
? Feld%(0);Feld%(1);Feld%(2)
```

Im ersten Element des Datenfelds steht die Breite des Grafikobjekts, im zweiten die Höhe und im dritten die Anzahl an Bitebenen. An diesen Daten dürfen Sie aber nichts verändern, sonst kommt AmigaBASIC mit der internen Berechnung der Grafikdaten durcheinander.

Zum PUT-Befehl gibt es noch etwas zu erklären: Sie können hinter dem Feldnamen noch eine dritte Angabe machen. Nämlich eine logische Verknüpfung, die AmigaBASIC zwischen der Grafik aus dem Datenfeld und dem Bildschirmhintergrund durchführen soll. Normalerweise verknüpft AmigaBASIC die Grafik und den Hintergrund mit XOR: Wo im Hintergrund keine Punkte gesetzt sind, erscheint ein Grafikpunkt in normaler Darstellung. Wo im Hintergrund bereits Punkte waren, wird invertiert. Ein Versuch im Direktmodus beweist es:

```
cls
line (100,120)-(199,139),1,bf
put (100,100),Feld%
```

Die Teile des Gesichts, die den gezeichneten Block überlagern, werden invertiert. Ein interessanter Effekt tritt ein, wenn Sie denselben PUT-Befehl noch mal eingeben:

```
put (100,100),Feld%
```

Da staunen Sie, nicht wahr? Das Gesicht ist völlig verschwunden. Durch die XOR-Verknüpfung wird eine Grafik durch einen zweiten PUT-Befehl an derselben Stelle wieder vom Bildschirm entfernt, ohne daß der Hintergrund beschädigt wird. Diesen Trick könnten Sie auch für Animationseffekte verwenden. Aber dafür gibt es schließlich die Bobs. Und die funktionieren im Prinzip ganz genauso.

Probieren wir mal, was es sonst noch so gibt:

```
cls
put (100,100),Feld%,preset
```

Die Verknüpfung PRESET bewirkt eine invertierte Wiedergabe der Grafik. Werfen wir einen Blick auf die Farben bei invertierter Darstellung. Die vier Standardfarben der Workbench werden nach einer bestimmten Regel umgesetzt:

blau wird orange
weiß wird schwarz
schwarz wird weiß
orange wird blau

Die gleiche Umsetzung können Sie auch sehen, wenn Sie ein Icon auf der Workbench aktivieren. Es wird ja zur Kennzeichnung auch invertiert.

```
color 0,1
cls
put (100,100),Feld%,pset
```

Die Verknüpfung PSET ist die einfachste: Sie setzt die Grafik so auf den Bildschirm, wie sie abgespeichert wurde. Ohne Rücksicht auf den Hintergrund.

```
cls
line (100,120)-(199,139),1,bf
put (100,100),Feld%,and
```

In diesem Beispiel werden die Punkte mit dem Hintergrund AND-verknüpft: Nur wo vorher schon ein Punkt war, erscheint ein Punkt der Grafik.

Und was mit AND geht, muß mit OR auch gehen:

```
cls
line (100,120)-(199,139),1,bf
put (100,100),Feld%,or
```

Die OR-Verknüpfung kopiert die Grafik in den Bildschirm, ohne die Punkte bei Überdeckung zu invertieren. Schon steckt unser Männchen seine Nase über die Mauer.

Jetzt kennen Sie alle Verknüpfungsarten zwischen Grafik und Hintergrund. Je nach dem gewünschten Effekt können Sie die eine oder die andere benützen. Und Sie wissen, wie man einen Grafikausschnitt kopieren und an beliebiger Stelle im Bild darstellen kann. Doch mit alledem sind wir noch nicht viel weiter gekommen auf unserer Suche nach einer Methode, um Grafikdaten auf Diskette abzuspeichern.

Eine recht erfolgversprechende Idee wäre doch, die Elemente aus dem Datenfeld auf Diskette abzuspeichern. Schreiben Sie unser Beispielprogramm bitte ein wenig um:

```
DIM Feld%(563)

CIRCLE (50,20),40
CIRCLE (35,12),5
CIRCLE (65,12),5
CIRCLE (50,20),8
CIRCLE (50,18),30,,4,6
PAINT (50,20),3,1

GET (0,0)-(99,39),Feld%

OPEN "Männchen" FOR OUTPUT AS 1
  FOR x=0 TO 563
    PRINT #1,MKI$(Feld%(x));
  NEXT x
CLOSE 1
```

Wenn Sie selbst tippen, speichern Sie diese Version bitte unter einem eigenen Namen ab. Auf der beiliegenden Diskette finden Sie das Listing als "GetPutOutput" in der "Daten"-Schublade.

Das einzige, was Sie wahrscheinlich noch nicht kennen, ist die Funktion MKI$. Diesen Befehl werden Sie in Zukunft sicher sehr häufig brauchen. Er ist eine große Hilfe, um Zahlen platzsparend auf Diskette abzuspeichern. Dahinter steckt eine einleuchtende Überlegung: Wenn eine Zahl wie -32768 in gewohnter Form abgespeichert würde, bräuchte man dazu 7 Bytes. Ein Byte geht für das Minuszeichen verloren, fünf Bytes für die

Ziffern und mindestens ein Byte wäre als Trennzeichen zur nächsten Zahl nötig. Dabei wissen wir, daß die Zahl -32768 als 16-Bit-Zahl dargestellt werden kann. Demnach wäre sie 2 mal 8 Bits, also 2 Bytes groß. Wir könnten also mit weniger als einem Drittel des Speicherplatzes auskommen. Auch der Platz auf Diskette ist kostbar, aber das ist bei 880 KByte Speicherkapazität pro Diskette das geringere Problem. Vor allem die Schreib- und Lesezeiten können auf diese Weise drastisch verringert werden.

Um aus einer 16 Bit langen Integerzahl zwei Bytes zu machen, gibt es die Funktion MKI$ ("Make Integer String" = Wandle eine Integerzahl in einen String um). Es entsteht ein 2 Zeichen langer String, den wir dann auf Diskette schreiben. Übrigens: Für 8-Bit-Zahlen kennen Sie schon eine vergleichbare Funktion: Den Befehl CHR$. Mit ihm können Sie aus den drei Zeichen 255 ein Zeichen machen: CHR$(255). Beim Einlesen brauchen wir eine Funktion, die aus den beiden Bytes (bei MKI$) oder dem einen Byte (bei CHR$) wieder die zugehörige Zahl macht. Für CHR$ ist die Gegen-Funktion der Befehl ASC. Er gibt den ASCII-Code eines Zeichens an. Probieren Sie's ruhig einmal aus:

```
a$=chr$(10) : ? asc(a$)
```

Wäre 'a$' länger als ein Zeichen, würde ASC nur das erste Zeichen des Strings untersuchen. Die Funktion ASC ist auch hilfreich, wenn Sie mal auf die Schnelle den ASCII-Code eines Zeichens feststellen wollen:

```
? asc("H")
```

ergibt 72, den ASCII-Code des Großbuchstabens H. Die Umkehr-Funktion zu MKI$ heißt CVI. ("Convert to Integer" = Wandle in eine Integerzahl um)

```
a$=mki$(32000) : ? cvi(a$)
```

Der angegebene String muß mindestens zwei Zeichen lang sein. Längere Strings werden nur bis zur zweiten Stelle untersucht.

Um gleich Verwechslungen vorzubeugen, vergleichen wir ein paar Befehle, die auf den ersten Blick ähnliche Funktionen haben. Sie haben bereits vor einiger Zeit (im Kapitel 1.18) die Funktion CINT kennengelernt. Die dient dazu, eine normale Zahl durch Rundung in eine Integerzahl umzuwandeln:

`? CINT (3.9)` ergibt 4

Um eine ausgeschriebene Zahl, die in einem String steht, in eine Zahl umzuwandeln, gibt es den Befehl VAL.

`? VAL ("32768")` ergibt 32768

CVI hingegen macht aus zwei Bytes, die in einem String stehen, eine 16-Bit-Zahl und gibt deren Wert als Ergebnis aus.

```
a$=CHR$(0)+CHR$(254) : ? cvi(a$)
```

Die zwei Bytes erzeugen wir hier durch zwei einzelne CHR$-Befehle. Das Ergebnis wird 254 sein, da die höherwertigen 8 Bits alle den Wert 0 haben.

Wir wissen, daß das alles ein bißchen komplizierter wird, aber Sie stecken ja nun mittlerweile auch schon in den tiefsten Geheimnissen der BASIC-Programmierung. Und dazu gehören nun mal solche Zahlenkunststücke. Wenn Sie die Unterschiede verstanden haben, können wir daran gehen, unsere Grafikdaten wieder von Diskette einzulesen. Sie sparen sich viel Tipparbeit, wenn Sie das ursprüngliche Beispielprogramm laden und dort die nötigen Änderungen vornehmen. Oder natürlich, wie so immer, unsere Diskette im Buch benutzen und das Programm "Get-PutInput" laden.

```
DIM Feld%(563)

OPEN "Männchen" FOR INPUT AS 1
  FOR x=0 TO 563
    Feld%(x)=CVI(INPUT$(2,1))
  NEXT x
CLOSE 1
```

```
CLS
FOR x=0 TO 5
  FOR y=0 TO 4
    PUT (x*100,y*40),Feld%
  NEXT y
NEXT x
```

Na also! Wenn Sie das Programm schon ausprobiert haben, haben Sie gemerkt: Es erscheinen wieder unsere Männchen auf dem Bildschirm, ohne daß wir einen einzigen CIRCLE-Befehl im Programm verwendet hätten. Der Inhalt von 'Feld%' wurde von Diskette eingelesen.

CVI haben wir Ihnen ja ausführlich erklärt, aber vielleicht haben Sie noch ein paar Zweifel wegen INPUT$. Diesen Befehl haben wir schon häufig verwendet, ohne ihn bisher genauer zu erklären. Also holen wir mal das Versäumte nach.

Sie verwenden INPUT$, wenn Sie eine bestimmte Anzahl an Zeichen aus einer Diskettendatei lesen wollen. INPUT$(2,1) liefert einen String, der aus den nächsten beiden Zeichen in der Datei Nummer 1 besteht. Die Syntax lautet also:

INPUT$ (Anzahl Zeichen, Dateinummer)

Da Sie vorher festlegen, wieviele Zeichen Sie einlesen wollen, braucht INPUT$ auch keine Trennzeichen. Für CVI benötigen wir zwei Bytes, also lesen wir auch zwei Stück aus der Datei.

Sie müssen nur aufpassen, daß Sie nicht mehr Zeichen aus der Datei lesen wollen, als Zeichen vorhanden sind. Sonst meldet AmigaBASIC einen "Input past end"-Error - "Sie wollten Daten über das Dateiende hinaus einlesen". Um das zu vermeiden, gibt es zwei Möglichkeiten:

Zum einen die Funktion LOF(x). Sie steht für "Length of File", deutsch: Dateilänge. Die Zahl in Klammern gibt die Dateinummer an. Solange eine Datei geöffnet ist, ergibt LOF(Dateinummer) die Länge der Datei in Bytes. Sie können soviele Zeichen lesen, wie LOF(x) Ihnen mitteilt. Erinnern Sie

sich noch an das Einlesen von Grafikobjekten? Da hieß es: INPUT$(LOF(1),1). Alle Zeichen aus der Datei 1 werden in einen String gepackt. Wenn Sie einzelne Zeichen lesen, benutzen Sie vielleicht eine Schleife: FOR x=1 TO LOF(1). Zum anderen gibt es die Funktion EOF(x). Die kennen Sie schon. Sie liefert -1, wenn die Datei zu Ende ist, sonst den Wert 0. Das können Sie auch für Schleifen nutzen: WHILE EOF(1)=0, oder noch eleganter und in fast lupenreinem Englisch: WHILE NOT EOF(1).

In unserem Fall stellten sich solche Probleme nicht, denn wir wußten ja, wieviele Zeichen wir in die Datei geschrieben hatten.

Zuletzt noch ein interessanter Nebeneffekt von INPUT$: Wenn Sie keine Dateinummer angeben, liest INPUT$ die gewünschten Zeichen von der Tastatur ein. Probieren Sie:

```
? input$(10)
```

Nach dem Drücken von <RETURN> erscheint der BASIC-Cursor, und nichts weiter tut sich. Geben Sie ein paar Zeichen ein. Die Zeichen werden nicht angezeigt, aber der Amiga merkt sie sich. Nach dem 10. Zeichen erscheint der ganze String auf dem Bildschirm.

Auch beim Datenaustausch mit anderen Peripheriegeräten oder zur Datenfernübertragung ist INPUT$ hervorragend geeignet. Er ist in AmigaBASIC wohl das wichtigste Hilfsmittel, um Daten von irgendwoher einzulesen.

Sie haben jetzt eine Methode kennengelernt, Grafikdaten auf Diskette abzuspeichern. Für Bildausschnitte ist diese Methode auch gar nicht schlecht. Wenn es um ganze Bilder geht, stößt sie allerdings an ihre Grenzen. Für ein Bild aus unserem Malprogramm mit 320 * 200 Punkten und 5 Bit-Ebenen bräuchten Sie ein Datenfeld mit 21000 Elementen. So viel bekommen Sie nie auf einmal im Speicher von AmigaBASIC unter. Sie müßten also das Bild stückchenweise mit GET einlesen und dann abspeichern. Das ist kompliziert und zeitaufwendig. Aber es gibt eine andere Möglichkeit, die Bilder auf Diskette zu bekommen. Sie ist viel eleganter und bietet noch andere, ungeahnte Vorteile.

4.2 Das Daten-Jet-set - Interchange File Format (IFF)

Sofern Sie zwei oder mehrere kommerzielle Grafikprogramme besitzen, vielleicht sogar Programme wie "Deluxe Video" oder den "Aegis Animator", ist Ihnen sicher eines aufgefallen: Die meisten Amiga-Grafikprogramme sind datenkompatibel. Sie können mit "Graphicraft" ein Bild malen und dieses Bild von "DeluxePaint" einlesen lassen, Sie können die entstandenen Bilder oder Bildteile sogar als Hintergrund oder Akteure in einem der Animationsprogramme verwenden. Und vieles mehr.

Diese erfreuliche Tatsache ist keineswegs selbstverständlich. Bei den meisten Computern haben sich die Software-Firmen nicht so gut abgesprochen. Das Resultat ist dort eine ausgeprägte Eigenbrötlerei, die es völlig unmöglich macht, die Daten, die mit Programm A erzeugt wurden, auch nur ansatzweise mit Programm B zu bearbeiten. Besitzer des Commodore 64 haben hier zum Beispiel leidvolle Erfahrungen machen müssen.

Daß das auf dem Amiga so schön klappt, haben wir der amerikanischen Softwarefirma Electronic Arts zu verdanken. Die Leute von Electronic Arts haben sich von Anfang an sehr stark für den Amiga engagiert.

So hat diese Firma im Jahr 1985 ein Konzept erarbeitet, wie man Daten in eine einheitliche Form bringen kann, so daß sie von jedem Programm verstanden werden, ohne aber das einzelne Programm in seiner Leistung und seiner zukünftigen Entwicklung einzuengen. Das ganze Ding ist schlichtweg genial. Und damit das Kind einen Namen hat, nannte man das Resultat "Interchange File Format" (Austausch-Dateiformat) oder kurz IFF. In langwieriger Überzeugungarbeit haben die Entwickler von IFF vielen anderen Softwarefirmen (also immerhin der Konkurrenz) die Vorteile einer Vereinheitlichung nahegelegt. Und das mit guten Argumenten: Schließlich hat nicht nur der Kunde einen großen Vorteil davon, auch die Softwarefirmen verbessern ihre Absatzchancen. Der Nutzwert eines Programms steigt beträchtlich, wenn es seine Leistungen in ein vorhandenes Gesamtsystem einbinden kann: 100 $ für ein Malprogramm, mit dem man nichts weiter tun kann als Malen, ist ein hoher Preis.

100 $ für ein Programm, mit dem man malen, Videohintergründe erzeugen, Animationsobjekte bearbeiten und Geschäftsgrafiken interessanter gestalten kann, sind schon sehr viel besser angelegt.

Der IFF-Standard ermöglicht jederzeit zukünftige Erweiterungen, ohne daß die alten Dateien deshalb nicht mehr gelesen werden können. Die Daten sind so gekennzeichnet, daß sich jedes Programm die Teile herauspicken kann, die es versteht. Was es nicht versteht, überliest es einfach.

IFF ist nicht allein für Grafiken gedacht. Auch Noten, Musikinstrumente, Texte, Schriften etc. können in diesem Format abgespeichert werden.

Wir wollen uns aber ausschließlich mit den Grafikdaten beschäftigen. Wenn man auf dem Amiga Grafikprogramme schreibt, gehört es nicht nur zum guten Stil, die Datenein- und -ausgabe IFF-kompatibel zu halten, es bietet auch handfeste Vorteile: Es ist jederzeit möglich, die Bilder professioneller Malprogramme in AmigaBASIC zu übernehmen. Und es ist auch möglich, die Bilder des BASIC-Malprogramms in die professionellen Grafikprogramme einzulesen und deren Features zu nutzen. Zum Beispiel für erweiterte Hardcopy-Möglichkeiten oder für das Ausschneiden und Verschieben von Bildteilen. Kurz, für alle Möglichkeiten, die das entsprechende Programm bietet.

Und wer auch nur irgendein IFF-kompatibles Malprogramm besitzt, kann von diesem Datenaustausch Gebrauch machen.

Bevor wir ans Programmieren gehen, wollen wir Ihnen die wichtigsten Grundlagen und die Funktionsweise von IFF kurz erklären.

Daten, die alle dieselbe Funktion haben, werden in IFF zu einer Gruppe zusammengefaßt. Der Fachausdruck für so eine Gruppe heißt "Chunk". Das ist eigentlich eine ziemlich respektlose Bezeichnung, die übersetzt soviel wie "Haufen" oder "Klumpen" bedeutet. (Es gibt übrigens auch diese Restaurants, die

sogennantes junk food verkaufen. Das ist etwas gehacktes Fleisch inmitten eines ehemaligen Brötchens. Bitte nicht verwechseln.)

Zurück zu IFF: Eine IFF-Datei besteht aus mehreren Chunks. Alle Chunks, die zusammengehören, bilden gemeinsam eine "Form", was auf Deutsch auch nichts anderes heißt als "Form". Am treffendsten übersetzt man das englische Wort hier mit "Figur" oder "Formular".

Damit sich die Programme beim Lesen und Schreiben zurechtfinden, steht in jedem Chunk und in jeder Form zu Beginn eine Kennung, worum es sich handelt und wie lang das Gebilde ist. Eine IFF-Grafikdatei besteht aus mindestens drei Chunks. In einem stehen Steuerdaten, wie etwa die Breite und die Höhe des Bildes und die Anzahl der Bitebenen.

In einem weiteren befinden sich die RGB-Werte der Farben. Und im dritten Chunk steht die eigentliche Grafik. Oder genauer: Hier sind die Bits gespeichert, aus denen sich die Grafik zusammensetzt. Eine Ansammlung von Bits, die eine Grafik bilden, heißt "Bitmap" (Bit-Landkarte). Diese Bezeichnung paßt ja ganz gut zu den Bit-Ebenen, die Sie schon kennen: Die Bit-Ebenen sind in der Bit-Landkarte verzeichnet. Wir wollen aber ab jetzt statt Bit-Landkarte lieber Bitmap sagen. Es können noch weitere Chunks dazukommen, etwa die Daten über eine Farbpalettenanimation (das haben Sie sicher schon gesehen: Durch zyklische Farbänderungen vermittelt ein Bild die Illusion einer Bewegung) oder Daten über bestimmte Betriebsarten der Video-Hardware oder die Koordinaten, wo ein einzeln gespeicherter Bildausschnitt auf dem Bildschirm erscheinen soll.

Die Kennzeichnung eines Chunks oder einer Form besteht aus vier Bytes. Die vier Bytes bilden ein Codewort, aus dem eine IFF-Leseroutine erkennen kann, worum es sich bei dem aktuellen Gebilde handelt. Danach folgen vier weitere Bytes, in denen gespeichert ist, aus wievielen Daten das Bild besteht.

Diese Skizze zeigt den Aufbau einer typischen IFF-Grafikdatei:

```
FORM                     40156
ILBM
  BMHD                   20
  320, 200, 0, 0, 3, 0, 0, 0, 0, 10, 11, 320, 200
  CMAP                   96
  32, 48, 160, 240, 240, 240, 0, 0, 0, 240, 128 , ...
  BODY                   40000
  124, 142, 42, 213, 140, 0, ...
```

Bild 14: Der Aufbau einer IFF-Datei

Die ersten vier Zeichen in der Datei enthalten das Wort FORM. Die nächsten vier Bytes enthalten als 32-Bit-Zahl die Länge der Form, nämlich 40156 Bytes. Das Leseprogramm weiß jetzt: Ich habe den Anfang einer Form gelesen, die insgesamt 40156 Bytes lang ist. Danach folgt eine Kennung, welche Art von Daten die Form eigentlich beinhaltet. Hier lesen wir in unserem Beispiel die Bezeichnung ILBM. Das steht für "Interleaved Bitmap" (Eingebundene Bitmap - die Bitmap ist zusammen mit anderen Daten in eine Form eingebunden). Das Leseprogramm weiß jetzt: Die restlichen Daten dieser Form enthalten Chunks, die alle irgendetwas mit einer Grafik zu tun haben.

Und schon kommt auch der erste Chunk: Vier Bytes beinhalten die Kennung BMHD ("Bitmap Header" - Bitmap-Überschrift, gemeint sind die Steuerdaten). Die darauf folgenden vier Bytes ergeben die Zahl 20: Der BMHD-Chunk besteht aus 20 Bytes. Das Leseprogramm kann nun zu dem Unterprogramm verzweigen, das die Steuerdaten liest und interpretiert. Danach lesen wir die vier Zeichen CMAP aus der Datei. Sie weisen auf eine "Color Map" (eine Farb-Tabelle) hin, die... Na? Genau, die 96 Bytes lang ist. Dieser Chunk enthält Daten über die RGB-Zusammensetzung der Farben. Für R, G und B jeweils ein Byte, macht für 32 Farben nach Adam Riese 96 Bytes. Also können jetzt vom zuständigen Unterprogramm die Farben eingelesen werden.

96 Bytes später lesen wir aus der Datei BODY, 40000 Bytes lang. Der Chunk "Body" ("Körper, Rumpf, Hauptteil") beinhaltet nicht etwa die Maße von Raquel Welch oder Bo Derek, sondern die Bitmap der Grafik. Wie die Daten hier angeordnet sind, ist vom IFF-Standard genormt: Die Bildzeilen der verschiedenen Bitebenen stehen hintereinander in der Datei. Zuerst kommt die erste Zeile der ersten Bitebene, dann die erste Zeile der zweiten Bitebene usw. Sind alle Bitebenen durch, folgt die zweite Zeile der ersten Ebene, die zweite Zeile der zweiten Ebene und so weiter.

Wenn das Leseprogramm das letzte Byte der letzten Zeile aus der letzten Bitebene gelesen hat, hat es damit auch das 40156. Byte gelesen: Die IFF-Datei ist zu Ende.

Es gibt übrigens keine vorgeschriebene Reihenfolge für die Chunks. Theoretisch könnte auch zuerst BMHD, dann BODY und dann CMAP kommen. Die Steuerdaten BMHD müssen allerdings auf jeden Fall zuerst in der Datei stehen, damit das Leseprogramm weiß, welche Ausmaße und welche Anzahl an Bitebenen für die Bitmap im BODY-Chunk gelten. Die Reihenfolge BMHD - CMAP - BODY ist am sinnvollsten. Zuerst braucht das Programm die Steuerdaten, dann kann es die Farben lesen und einstellen, wodurch das Bild beim Einlesen schon in den richtigen Farben auf dem Bildschirm gezeigt wird.

Ein Hinweis noch, dann kommen wir zum praktischen Teil: Ein Chunk muß grundsätzlich aus einer geraden Anzahl von Bytes bestehen. Beinhaltet der CMAP-Chunk z.B. nur Farbwerte für 7 Farbregister (21 Bytes), muß er durch ein Füllbyte, meistens eine 0, ergänzt werden. Dieses Füllbyte wird in der Längenangabe des Chunks nicht mitgezählt. Ungerade Werte für die Anzahl müssen wir also um 1 (auf die nächste gerade Zahl) erhöhen. Das sieht in der Skizze so aus:

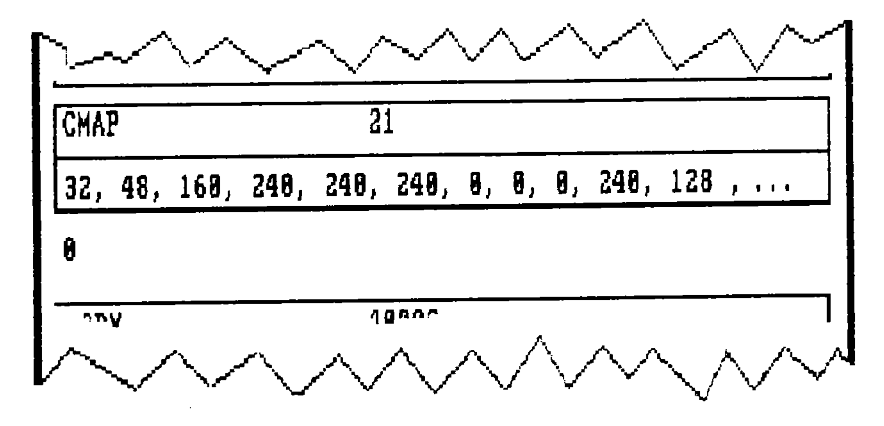


Bild 15: Ungerade Chunks werden durch ein Füllbyte ergänzt

4.3 Her mit den kleinen Amerikanern - die IFF-Leseroutine

Jetzt geht es nur noch darum, unsere theoretischen Vorkenntnisse über IFF so einzusetzen, daß wir IFF-Daten - eine Glanzleistung amerikanischer Programmierkunst - einlesen können.

Bevor wir daran gehen, noch eine Anmerkung: Seit einiger Zeit liefert Commodore eine Version der Extras-Diskette aus, auf der sich in der Schublade "BASICDemos" drei Programme befinden, die Ihnen ebenfalls ermöglichen, IFF-Bilder einzulesen und abzuspeichern. Diese Routinen wurden unter Verwendung von "Libraries" (über die erfahren Sie mehr im Anhang B) geschrieben und sind deshalb wesentlich schneller als die Routinen, die wir Ihnen in den folgenden Kapiteln vorstellen werden. Die Programme auf der Extras-Diskette sind allerdings wesentlich kom-

plizierter zu verstehen und anzuwenden. Aus diesem Grund meinen wir, daß es sich nach wie vor lohnt, bei der Programmierung unserer IFF-Programme mitzumachen, die ausschließlich in AmigaBASIC und ohne den Einsatz von Libraries geschrieben wurden. Wir haben ja schon vorher gesagt, daß die Beispiele in unserem Buch Ihnen einen praktischen Nutzen bringen, aber vor allem die Anwendung von AmigaBASIC-Befehlen erklären sollen.

Da Sie in den folgenden Beispielen nicht nur viel über die Programmierung von Diskettenein- und -ausgabe und andere Möglichkeiten von AmigaBASIC erfahren, sondern auch den Aufbau und die Funktionsweise von IFF-Dateien verstehen lernen sollen, raten wir Ihnen, die Programme trotz unserer Commodore-Konkurrenz mit abzutippen. Natürlich können Sie sie auch von der beiliegenden Diskette laden (Name: "IFF.Read", Schulade: "Daten"). Nur beschäftigen sollten Sie sich in jedem Fall damit. Was Sie über die neuen Beispielprogramme "LoadACBM", "LoadILBM-SaveACBM" und "SaveILBM" wissen müssen, erfahren Sie im Anhang C. Unsere Erkenntnisse und Möglichkeiten haben das nächste Programm hervorgebracht:

```
INPUT "Dateiname";Nam$
OPEN Nam$ FOR INPUT AS 1
  Form$=INPUT$(4,1)
  Laenge=CVL(INPUT$(4,1))
  IF INPUT$(4,1) <> "ILBM" THEN
    PRINT "Fehler im Dateiformat!"
    END
  END IF

Lesen:
  IF EOF(1) THEN END
  Chunk$=INPUT$(4,1)
  Laenge=CVL(INPUT$(4,1))
  IF INT (Laenge/2)<>(Laenge/2) THEN Laenge=Laenge+1
  IF Chunk$="BMHD" THEN BMHD
  IF Chunk$="CMAP" THEN CMAP
```

```
  IF Chunk$="BODY" THEN BODY
  Dummy$=INPUT$(Laenge,1)
GOTO Lesen

BMHD:
  xd=CVI(INPUT$(2,1))
  yd=CVI(INPUT$(2,1))
  Dummy$=INPUT$(4,1)
  Biteb=ASC(INPUT$(1,1))
  Dummy$=INPUT$(11,1)
  IF xd=320 THEN Typ=1
  IF xd=640 THEN Typ=2
  IF yd>200 THEN Typ=Typ+2
  SCREEN 1,xd,yd,Biteb,Typ
  WINDOW 2,Nam$,,0,1
  Ad=PEEKL(WINDOW(8)+4)+8
  FOR x=0 TO Biteb-1
    EbAd(x)=PEEKL(Ad+4*x)
  NEXT x
  Dummy$=INPUT$(Laenge-20,1)
GOTO Lesen

CMAP:
  FOR x=1 TO Laenge/3
    r=(ASC(INPUT$(1,1)) AND 240)/16
    g=(ASC(INPUT$(1,1)) AND 240)/16
    b=(ASC(INPUT$(1,1)) AND 240)/16
    PALETTE (x-1),r/16,g/16,b/16
  NEXT x
  IF INT(Laenge/3)<>(Laenge/3) THEN Dummy$=INPUT$(1,1)
GOTO Lesen

BODY:
  BytesZeile=xd/8
  FOR y1=0 TO yd-1
    FOR b=0 TO Biteb-1
      FOR x1=0 TO BytesZeile/4-1
        POKEL EbAd(b)+4*x1+BytesZeile*y1,CVL(INPUT$(4,1))
```

```
            NEXT x1
          NEXT b
        NEXT y1
      GOTO Lesen
```

Dieses Programm ist in der Lage, IFF-Bilder in BASIC einzulesen. Probieren Sie's ruhig gleich mal aus. Wenn Sie das Programm selbst eingetippt haben, sollten Sie es aber unbedingt vorher abspeichern, denn es gibt ein paar kritische Stellen, an denen ein einziger Tippfehler einen Systemabsturz verursachen könnte. Dann wäre die ganze Tipparbeit umsonst gewesen.

Starten Sie das Programm, geben Sie als Dateinamen den Namen eines Bilds von einer Ihrer Grafik-Disketten ein, z.B. "Graphicraft:Skier", legen Sie die jeweilige Diskette ins Laufwerk, und los geht's. Sollten Sie überhaupt keine Grafikdiskette besitzen, schreiben Sie einfach "ExtrasD:BasicDemos/Heart.ILBM".

Das Programm legt einen Screen in der benötigten Auflösungsstufe an, liest die Farben ein (Sie sehen das z.B. daran, das sich die Farben des Mauscursors ändern.) und baut dann Zeile für Zeile das Bild auf. Beachten Sie, was mit dem Window-Rahmen passiert: Beim Einlesen wird er von den Bilddaten überschrieben. Erst wenn Sie die Menütaste der Maus drücken, erscheint der Rahmen wieder. Passen Sie auf, daß Sie keine Pulldowns herunterziehen, solange die obere Hälfte des Bilds geladen wird. Die eingelesenen Grafikdaten überschreiben sonst den Pulldown-Inhalt und auf dem Grafikbildschirm fehlt dafür der Teil, der durch das Pulldown verdeckt wurde.

Das Ergebnis ist schon toll: Sie haben ein BASIC-Window, in dem eine IFF-Grafik dargestellt wird. Damit bieten sich jetzt tausend Möglichkeiten: Sie können Text über der Grafik darstellen, Teile der Grafik mit GET und PUT kopieren, Grafikobjekte vor dem Bild bewegen und vieles mehr. Einige Anregungen und praktische Beispiele werden wir Ihnen vorstellen. Aber lassen Sie ruhig ein wenig Ihre Phantasie spielen, was man sonst noch alles machen könnte.

Allerdings noch ein Hinweis, falls Sie bei Ihrem Experiment mit dem Programm eine Enttäuschung erlebt haben sollten: Schwierigkeiten kann es zum Beispiel geben, wenn Sie versuchten, ein mit "Deluxe Paint" erstellten Bild einzulesen. Woran das liegt, und was man dagegen tun kann, erfahren Sie im Zwischenspiel 5.

Es gibt sicher auch noch ein paar Fragen zum Programm selbst. INPUT verlangt einen Dateinamen, die angegebene Datei wird zum Lesen geöffnet. Der erste INPUT$(4,1) liest die Kennung FORM ein, in die Variable 'Laenge' kommt die Form-Länge. Diese Daten werden wir nicht mehr brauchen.

Der Befehl CVL in der vierten Zeile muß wirklich CVL heißen, wir haben uns nicht vertippt. So wie CVI 2 Bytes in eine 16-Bit-Zahl umwandelt, wandelt CVL ("Convert Long") 4 Bytes in eine 32-Bit-Zahl. Der zugehörige Befehl beim Schreiben heißt übigens MKL$. Warum es nun nötig war, gleich 32 Bits für die Länge einer Form zu veranschlagen, können wir Ihnen auch nicht sagen. Auf jeden Fall ermöglicht der IFF-Standard auf diese Weise Dateilängen bis zu 4.294.967.295 Bytes Länge. Bis wir Zeiten erleben dürfen, in denen irgendjemand 4 GigaByte mit Grafikdaten füllt, wird es aber wohl noch etwas dauern.

Die nächsten vier Bytes der Datei sollten die Kennung ILBM enthalten. Ist das nicht der Fall, lesen wir keine ILBM-Datei, das Programm bricht ab.

In die Variable 'Chunk$' wird die Chunk-Kennung gelesen, in 'Laenge' die Chunk-Länge.

Mit der Zeile IF INT(Laenge/2)<>(Laenge/2)... können wir überprüfen, ob 'Laenge' eine gerade Zahl beinhaltet: Wir teilen 'Laenge' durch 2 und schneiden bei einem der Ergebnisse den Nachkommateil ab. Sind die beide Werte gleich, gibt es keinen Nachkommateil, die Zahl ist also durch 2 teilbar. Die Zeichen <> bedeuten "ungleich". Bei einer ungeraden Zahl wird die Zahl um 1 erhöht, dann ist sie gerade.

Je nach dem Inhalt von 'Chunk$' verzweigen wir zu dem zuständigen Programmteil. Wurde der Chunk nicht erkannt, lesen wir in 'Dummy$' seinen gesamten Inhalt ein. "Dummy" ist ein beliebter Name für einen Platzhalter, eine Attrappe. 'Dummy$' beinhaltet sozusagen unseren Datenabfall, nämlich die Daten, die das Programm nicht interpretieren kann.

Die Schleife 'Lesen:' wiederholen wir, bis keine Daten mehr in der Datei sind.

Im Programmteil 'BMHD:' lesen wir den gleichnamigen Chunk. Er hat standardmäßig 20 Bytes. Die ersten beiden Bytes beinhalten als 16-Bit-Zahl die Breite der Grafik, die nächsten beiden ihre Höhe. Wir weisen die Werte den Variablen 'xd' und 'yd' zu.

Die nächsten vier Bytes wandern in unseren 'Dummy$'. Nach dem IFF-Standard stehen hier für Bilder, die nicht den ganzen Bildschirm einnehmen, die X- und die Y-Koordinate der linken oberen Ecke. Wir wollen nur vollständige Bilder einlesen, deshalb interessieren uns die beiden Bytes nicht.

Das nächste Byte beinhaltet die Anzahl der Bitebenen, wir weisen den Wert der Variablen 'Biteb' zu.

Nun folgen noch 11 Bytes, die wir alle nicht brauchen. Was hier im einzelnen an Daten steht, werden Sie erfahren, wenn wir eine Routine zum Speichern im IFF-Format vorstellen.

'xd' kann den Wert 320 oder 640 haben. Und 'yd' beinhaltet entweder die Zahl 200 oder bei Interlace-Bildern die Zahl 400. Je nach Auflösungsstufe errechnen wir einen Wert für 'Typ', die Variable, die beim folgenden SCREEN-Befehl eine von vier Auflösungen angibt.

Auf dem erzeugten SCREEN legen wir ein Window an, das den Namen der eingelesenen Datei trägt.

Jetzt folgt der trickreichste Teil, für den wir wieder etwas ausholen müssen: Der Amiga muß, wie jeder Computer, ständig Übersicht über seine Speicherverwaltung behalten. Wo im Speicher wichtige Programme stehen (z.B. die Workbench-Routinen oder AmigaBASIC), dürfen natürlich keine Daten oder Bitmaps abgelegt werden. Durch das Multitasking des Amiga gibt es keine einheitliche Speicheraufteilung. Man kann nicht sagen, "Hier geht grundsätzlich AmigaBASIC los." - es kann bei anderen Vorbedingungen auch an einer ganz anderen Stelle im Speicher stehen.

Jede Speicherzelle kann genau 1 Byte speichern. Je nach Speicherausbau hat der Amiga für seinen Schreib-/Lesespeicher ungefähr 262000 oder 524000 solcher Speicherzellen zur Verfügung. Außer dem Schreib-/Lesespeicher (RAM) hat der Amiga auch noch 256 KByte Festwertspeicher (ROM), in dem sich der Inhalt der ehemaligen Kickstart-Diskette befindet. So ist es zumindest beim Amiga 500 und 2000. Der Amiga 1000 hat 256K RAM, in die beim Starten der Inhalt der Kickstart-Diskette geladen wird. Nach dem Laden wird das RAM elektronisch verriegelt, so daß es sich nach außen wie ROM verhält. Damit man eine Speicherzelle konkret ansprechen kann, hat jede Zelle eine Nummer. Diese Nummer nennt man auch Adresse. Die allererste Speicherzelle im Amiga hat die Nummer 0, die nächste die Nummer 1 und so weiter.

Es gibt im Amiga-*Betriebssystem* ein Programm, das ständig überwacht, welche Speicherbereiche von welchem Programm belegt sind. Dieses Programm weiß zum Beispiel: "Die Adressen von Nummer 0 bis Nummer 40000 werden von der Workbench benötigt. Die Adressen 40001 bis 72000 werden für die Bildschirmdarstellung benutzt. Von 72001 bis 160000 hat sich AmigaBASIC breitgemacht." Und so weiter. Die Zahlen stimmen übrigens nicht, sie sind nur als Bespiel gedacht. Das Speicherverwaltungs-Programm ist Teil von "Exec". Und "Exec" wiederum ist seinerseits ein Teil des Betriebssystems und für Aufgaben wie Speicherverwaltung und Multitasking zuständig.

Normalerweise sollten Sie die Inhalte von Speicherstellen nicht direkt verändern, da dabei meistens eher Schaden als Nutzen entsteht. Trotzdem bietet AmigaBASIC eine Reihe von Befehlen, um genau das zu ermöglichen: Mit PEEK können Sie den Inhalt einer Speicherzelle auslesen, mit POKE einen Wert in eine Speicherzelle schreiben. Für manche Computer sind diese Befehle sehr wichtig, weil sie dort die einzige Möglichkeit darstellen, nicht vorhandene BASIC-Befehle zu ersetzen. Beim Commodore 64 z.B. gibt es so viele nützliche PEEKs und POKEs, daß zwei Leute, die ein Buch darüber geschrieben haben, ziemlich viel Geld verdienen konnten. Beim Amiga brauchen Sie PEEK und POKE viel, viel seltener, weil es für fast alle Funktionen eigene BASIC-Befehle gibt. Aber zum Abspeichern und Laden von Grafiken bietet AmigaBASIC keine Befehle, also greifen wir zu PEEKs und POKEs. Alles klar?

Geben Sie im BASIC-Window ein:

```
? PEEK (30000)
```

Sie erhalten eine Zahl zwischen 0 und 255. Das ist der Wert des Bytes, das in der Speicherzelle mit der Adresse 30000 abgespeichert ist. Bildlich gesprochen: Sie erhalten einen Brief mit dem Absender "30000", dessen Inhalt eine 8-Bit-Zahl ist. Der Bewohner der Adresse 30000 hat Ihnen geschrieben, welche Zahl er sich gerade merkt. (Natürlich können wir verstehen, daß Ihnen ein Brief, sagen wir mal über eine hohe Steuerrückerstattung lieber ist. Aber bei Computern ist halt alles etwas trockener...)

POKE 30000, (Wert)

schreibt einen Wert in die Speicherzelle mit der Adresse 30000. Setzen Sie am besten den Wert ein, den Sie oben mit PEEK ausgelesen haben. Sie schicken einen Brief an die Adresse "30000", in dem Sie dem Bewohner der Speicherzelle die 8-Bit-Zahl mitteilen, die er sich ab jetzt merken soll.

Wollen Sie nicht 8, sondern 16 Bits lesen bzw. schreiben, verwenden Sie dazu die Befehle PEEKW und POKEW. (W für "Wort", den Ausdruck, den wir eigentlich nicht mehr für eine

16-Bit-Zahl verwenden wollten. Aber die Wege des Schicksals sind unergründlich.) Diese Befehle arbeiten mit zwei Bytes, die ab der angegebenen Adresse hintereinander im Speicher stehen. Die Bewohner der beiden Speicherzellen teilen sich die 16 Bits: Jeder merkt sich 8 Bits. Ihr Ansprechpartner ist in jedem Fall der Bewohner der Speicherzelle mit der geraden Nummer. Wenn Sie

POKEW 30001, (Wert)

eingeben, erhalten Sie die Fehlermeldung "Illegal function call".

Das ganze Spiel gibt es noch für 32-Bit-Zahlen. Die Befehle heißen hier PEEKL und POKEL (L wie "Langwort", Sie wissen ja, noch so ein Begriff...). Auch bei diesem Befehl muß die angegebene Adresse eine gerade Zahl sein. Bearbeitet werden die vier Speicherzellen, die ab der Adresse hintereinander im Amiga-Speicher liegen. Diesmal sind es vier kleine Speicherbewohner, jeder hat wieder nichts anderes als 8 Bits im Kopf - von insgesamt 32.

Dieses war der erste Streich. Doch der zweite folgt sogleich. Da gibt es die Funktion WINDOW(8).

WINDOW(0) kennen Sie schon. Es teilt Ihnen die Nummer des zur Zeit aktivierten Ausgabe-Windows mit. Doch es gibt noch viele andere WINDOW-Funktionen, mit den verschiedensten Aufgaben. WINDOW(8) gibt Ihnen die Adresse einer Liste ("Struktur" sagen die Fachleute und die C-Programmierer) an, in der sich Intuition die Daten des aktuellen Windows merkt. Das heißt, ganz so einfach ist's leider nicht. Der Weg zum Erfolg führt über einige Umwege: Die zweiten vier Bytes der Liste, auf die WINDOW(8) zeigt, beinhalten die Adresse, an der eine andere Liste beginnt. In dieser anderen Liste stehen ab dem 8. Byte mehrere 32-Bit-Werte: Das sind die Startadressen der einzelnen Bitebenen. Wenn Sie diese Schnitzeljagd im Amiga-Speicher nicht ganz, nur teilweise oder gar nicht verstanden haben, ist's auch nicht schlimm. Hauptsache, Sie verstehen, daß wir in

unserem Programm in dem Feld 'EbAd(x)' die Adressen abgelegt haben, an denen die einzelnen Bit-Ebenen unseres Grafik-Windows im Speicher liegen. Die brauchen wir nämlich später.

Zu guter Letzt liest unser Müllmann 'Dummy$' eventuell überzählige Bytes.

Das Lesen des CMAP-Chunks ist mit weniger Schwierigkeiten verbunden. Für jede Farbe werden drei Bytes gelesen: Eines für R, eines für G und eines für B. Daß der IFF-Standard auf die Zukunft ausgelegt ist, haben wir schon gemerkt. Dasselbe gilt auch für Farbabstufungen. Der Amiga kann für R, G und B jeweils 16 Stufen darstellen. IFF bietet die Möglichkeit, 256 Abstufungen pro Grundfarbe anzugeben. Um Bilder, die heute entstehen, auch auf zukünftigen, vielleicht leistungsfähigeren Computern verwenden zu können, haben sich die IFF-Entwickler einen schlauen Trick ausgedacht: Zum Abspeichern der Farbabstufungen von 0 bis 15 werden die obersten 4 Bits eines Bytes verwendet. Das bedeutet, zwischen jeder Farbabstufung des Amiga bleiben 15 Zahlen frei, die später für noch feinere Zwischenstufen genutzt werden können. Die prozentualen Anteile von Rot, Grün und Blau bleiben auch in Zukunft in der richtigen Größenordnung.

Beim Einlesen der Bytes isolieren wir die oberen vier Bits durch AND 240 und teilen das Ergebnis durch 16. So erhalten wir die richtigen Werte für den nachfolgenden PALETTE-Befehl. Bleibt am Schluß ein überzähliges Füllbyte übrig, nimmt sich 'Dummy$' seiner an.

Der letzte Programmteil liest den BODY-Chunk, also die Bitmap. Die Variable 'BytesZeile' errechnet aus 'xd' die Anzahl an Bytes pro Bildzeile. Die verschachtelten Schleifen lesen Bildzeile für Bildzeile. Im letzten Kapitel haben Sie gelernt, in welcher Reihenfolge die Zeilen abgespeichert werden: pro Bitebene eine Bildzeile. Genauso lesen wir sie wieder ein. Jetzt tritt unser mühsam erworbenes Datenfeld 'EbAd' in Aktion: Wir kennen die Startadressen der einzelnen Bitebenen und bringen Byte für Byte an die richtige Position im Speicher. Wir verwenden dazu

den Befehl POKEL, weil wir mit ihm die schnellsten Ergebnisse erzielen: Pro Schleifendurchlauf werden 32 Bits in den Speicher geschrieben.

Die Variable 'y1' zählt die Bildzeilen hoch, 'b' die Bitebenen und 'x1' die Nummer der einzelnen 32-Bit-Zahlen innerhalb der Zeile. Sind alle Daten der Bitmap in den Speicher übertragen, erfolgt der Rücksprung zur 'Lesen:'-Routine.

Sollten noch weitere Chunks in der Datei stehen, werden sie alle überlesen, da unser Programm sie nicht braucht oder die enthaltenen Daten sowieso nicht verstehen würde.

Es bleibt noch ein Problem zu besprechen, von dem Sie bei den Experimenten im letzten Kapitel vielleicht auch schon betroffen waren - besonders dann, wenn Sie das Programm "DeluxePaint" besitzen. Um dieses Problem und seine Lösung geht es im nächsten Zwischenspiel.

Zwischenspiel 5: Das deutsch-amerikanische Freundschafts-Projekt - Entzerren von komprimierten IFF-Dateien

Wenn Sie unser IFF-Leseprogramm eingetippt oder auch von der beigelegten Diskette geladen haben, haben Sie es sicher sofort ausprobiert. Sollten Sie dabei jedoch versucht haben, eine IFF-Datei einzulesen, die z.B. mit dem Programm "Deluxe Paint II" oder einem anderen professionellen Grafikprogramm erstellt wurde, mußten Sie wahrscheinlich eine Enttäuschung erleben.

Vielleicht erschienen da sehr wirre farbige Muster auf dem Bildschirm, die Ähnlichkeit mit so manchem haben könnten, aber sicher nicht mit dem Bild, das Sie laden wollten.

Woran liegt das? Nun, "Deluxe Paint", "Deluxe Paint II" und einige andere Grafikprogramme haben eine für unsere Zwecke ungünstige Eigenschaft: Sie speichern die Bitmap im BODY-Chunk in einem komprimierten Speicherformat ab. Der Grund dafür ist einfach, daß Grafiken meistens aus mehr oder weniger großen gleichfarbigen Flächen bestehen. Oft sind große Teile

des Bildschirms einfach mit der Hintergrundfarbe gefüllt, dort befinden sich keine weiteren Bilddetails. Einige Programmierer haben sich überlegt, daß es doch unsinnig ist, 2000mal denselben Wert hintereinander auf Diskette zu schreiben. Um Disketten-Speicherplatz zu sparen, haben sie eine Verschlüsselung entwickelt, durch die Folgen gleichwertiger Bytes gekennzeichnet werden. Ein Bild, das im Speicher 40 KByte braucht, nimmt dann auf Diskette vielleicht nur noch 25 KByte in Anspruch.

Für das Einlesen von Bildern in BASIC stellt die komprimierte Speicherung aber einen Nachteil dar. Eine Entzerr-Routine ist nämlich extrem aufwendig und beim Einlesen zeitraubend. Das bedeutet, daß Sie die Bilder, die von "Deluxe Paint" abgespeichert wurden, nicht direkt mit unserem "IFF.Read" einlesen können. Versuchen Sie's ruhig mal, falls Sie die Möglichkeit haben. Von dem ursprünglichen Bild ist jedenfalls nicht mehr viel zu erkennen. Außerdem bricht das Programm schließlich mit einem "Input past end"-Error ab, weil es in der Datei mehr Daten erwartet als in Wirklichkeit vorhanden sind.

Bei Bildern der Auflösungsstufe 320 * 200 ist die Sache nicht ganz so tragisch, wenn Sie z.B. das Grafikprogramm Graphicraft haben. Graphicraft entzerrt die Bilder beim Einlesen und speichert sie in jedem Fall unkomprimiert ab. Sie brauchen ein komprimiertes Bild also nur in Graphicraft einzulesen und wieder auf Diskette zurückzuspeichern, das war alles. Das Bild kann dann von BASIC aus problemlos eingelesen werden. Aber Graphicraft kann nicht in höheren Auflösungsstufen arbeiten. Bilder im 640 * 200-Modus oder sogar im 640 * 400-Modus werden von allen uns bekannten Grafikprogrammen komprimiert gespeichert. Diese Bilder könnten also von Ihnen trotz Graphicraft in BASIC nicht verwendet werden.

Und mit dem Programm "Graphicraft" hat es auch so seine Bewandtnis. Es wurde in der Anfangszeit des Amiga 1000 jedem Computer kostenlos beigelegt. So konnte man dann davon ausgehen, daß jeder Amiga-Besitzer das Programm besitzt. Heute ist das aber leider nicht mehr so. Spätestens seit der Einführung der Amigas 500 und 2000 ist Graphicraft so gut wie völlig vom

Markt verschwunden. Das ist auch nicht sehr verwunderlich, denn leistungsmäßig steht "Graphicraft" deutlich hinter Programmen wie "Deluxe Paint II" oder "Aegis Images" zurück.

Was also tun, wenn Sie nicht zufällig "Graphicraft" besitzen, oder wenn Sie Bilder im Format 640*200 oder 640*400 in BASIC einlesen wollen? In diesem Zwischenspiel werden wir Ihnen dafür eine Lösung vorstellen, die wirklich jedem zur Verfügung steht.

Sie alle besitzen nämlich ein anderes Programm, das in der Lage ist, komprimierte IFF-Dateien zu entschlüsseln. Die Rede ist von dem Programm "LoadILBM-SaveACBM" in der "BasicDemos"-Schublade auf Ihrer "ExtrasD"-Diskette.

Sie erinnern sich vielleicht, wir haben nämlich schon einmal darüber gesprochen: Mit diesem und zwei anderen Programmen, die allesamt von der amerikanischen Commodore-Mitarbeiterin Carolyn Scheppner geschrieben wurden, macht Commodore den Nachteil von AmigaBASIC wett, keine IFF-Dateien laden zu können. Lesen Sie dazu bitte auch die Anmerkungen zu diesen Programmen im Anhang C.

Die Programme sind wesentlich schneller als unser IFF-Ladeprogramm, weil sie mit Library-Routinen geschrieben sind, also Aufrufen von Teilen des Betriebssystems aus BASIC. Respekt vor der Leistung von Carolyn Scheppner. Das Prinzip, das in diesen Programmen verwendet wird, hat in unseren Augen jedoch einen gravierenden Nachteil. Vielleicht hat Sie der Programmname "LoadILBM-SaveACBM" schon ein wenig stutzig gemacht. Was eine ILBM ist, wissen Sie ja. Aber "ACBM"?

Im ILBM-Format liegen sämtliche IFF-Grafik-Dateien vor, die von irgendwelchen Amiga-Zeichenprogrammen erzeugt wurden. Dabei werden die Bilder zeilenweise gespeichert. Erst die Zeile 1 aller betroffenen Bitebenen, dann die Zeile 2 und so weiter. Der Vorteil dabei ist, daß Sie bereits beim Laden das Bild in den richtigen Farben sehen können. Und daß beim Einlesen in verschiedenen Auflösungsstufen dieses Format weniger Probleme macht. Wenn der Screen zu Ende ist, bevor das Bild komplett zu

sehen ist, dann fehlt eben ein Teil des Bildes. Aber der Rest wird immerhin richtig dargestellt. Und wenn der Screen zu groß ist, na dann bleibt eben ein Teil davon frei.

Beim Laden von ILBM-Dateien müssen jedoch eine ganze Menge Berechnungen durchgeführt werden, was die Ladezeit deutlich verlängert. Das merken Sie ja auch bei unserem Programm "IFF.Read".

Commodore hat aus diesem Grund einen neuen IFF-Dateityp eingeführt. Die sogenannten ACBMs. ACBM bedeutet "Amiga Continuous Bitmaps", auf deutsch: "Fortlaufende Amiga-Bitmaps". In dieser Speicherungsform werden die einzelnen Bit-Ebenen hintereinander abgespeichert, so wie sie auch im Amiga-Speicher stehen. Das Einlesen geht dann sehr schnell, da fast keine Rechenarbeit mehr anfällt. Der Geschwindigkeitsgewinn wird jedoch zu einem hohen Preis erkauft: ACBMs können mit vertretbarem Aufwand nur und ausschließlich auf dem Amiga in seiner gegenwärtigen Form geladen werden, und zwar nur in der Auflösungsstufe, in der sie abgespeichert wurden. Versucht man, diese Einschränkungen zu umgehen, wird der Rechenaufwand wesentlich größer, als wenn von vorneherein ILBMs eingesetzt worden wären.

Deshalb haben wir uns entschlossen, in unseren Beispielprogrammen weiterhin ausschließlich mit ILBMs zu arbeiten. Nichtsdestotrotz ist "LoadILBM-SaveACBM" ein tolles Programm, das unter anderem auch die Entzerrung der komprimierten Daten durchführen kann.

Was wir nun im Idealfall bräuchten, wäre ein Programm, das komprimierte ILBMs einliest und unkomprimierte ILBMs abspeichert. So ein Programm existiert (bisher) weder im Repertoire unserer Beispielprogramme aus diesem Buch noch auf der "ExtrasD" von Commodore. Aber die notwendigen Bestandteile existieren bereits: die schnelle, entzerrungsfähige ILBM-Laderoutine aus dem Programm "LoadILBM-SaveACBM" und ein IFF-Speicherprogramm auf unserer Diskette im Buch ("PicSave"). Unser Programm zum ILBM-Entzerren ist also sozusagen ein deutsch-amerikanisches Freundschaftsprojekt. Der amerikanische

Teil (LoadILBM-SaveACBM) wird von Mrs. Scheppner beigesteuert, und der deutsche Teil (die "Picsave"-Routine) stammt von uns.

Die "Picsave"-Routine zum Speichern von IFF-Daten haben wir zwar bisher noch nicht besprochen, wir wollen sie aber an dieser Stelle schon einsetzen, damit Sie so schnell wie möglich Ihre IFF-Grafiken entzerren können und dann die Möglichkeit haben, wirklich jedes Bild in BASIC einzulesen. Das Programm "Picsave" lernen Sie ausführlich in Kapitel 4.8 kennen. Die Arbeiten, die nötig sind, um die Programme "LoadILBM-SaveACBM" und "Picsave" zu unserem neuen Programm "ILBMEntzerren" zusammenzufügen, wollen wir Ihnen ersparen. Sie finden das fertige Programm in der "Daten"-Schublade auf der beiliegenden Diskette.

Dieses Programm ist dazu da, Ihnen zu helfen und das Problem mit den komprimierten IFF-Dateien zu lösen. Sie brauchen dieses Programm nicht unbedingt zu verstehen. Da dort unter anderem auch Libraries (Routinen aus dem Betriebssystem) zum Einsatz kommen, muß man schon ziemlich fortgeschritten in AmigaBASIC sein, um restlos durchzublicken.

Für alle Interessierten haben wir einen Überblick darüber zusammengestellt, was wir alles gemacht haben, um "ILBMEntzerren" zu erzeugen. Wer sich also näher mit dem Programm beschäftigen möchte, sieht hier, wie das Programm entstanden ist. Wenn Sie dieses Programm aber nur anwenden und sich mit den Details gar nicht oder erst später (z.B. wenn Sie die "Picsave"-Routine in Kapitel 4.8 kennengelernt haben) beschäftigen wollen, ist das auch in Ordnung.

Die folgenden Schritte zeigen Ihnen jedenfalls, wie wir bei der Erstellung des neuen Programms "ILBMEntzerren" vorgegangen sind.

- Wir haben den kompletten Programmteil "SaveACBM:" (bis einschließlich "Scleanup:") in "LoadILBM-SaveACBM" gelöscht und dann ans Programmende mit MERGE unsere "Picsave"-Routine angehängt.

- Die Kommentare und Texte im Programm haben wir, soweit nötig, geändert.

- Bevor in "BmapSuchen:" mit LIBRARY-Befehlen die drei Libraries "dos.library", "exec.library" und "graphics.library" aufgerufen werden, wechselt das Programm das aktuelle Verzeichnis mit CHDIR nach "ExtrasD:BasicDemos". Dort stehen die .bmap-Dateien, die der LIBRARY-Befehl benötigt. Nach dem Einlesen der Libraries kehren wir ins "BASICDisk:Daten"-Verzeichnis zurück.

  Was es mit .bmap-Dateien und Libraries auf sich hat, können Sie bei Interesse im Anhang B unter LIBRARY nachlesen. Da es dabei um so komplizierte Dinge wie die Einbindung von Betriebssystem-Routinen in BASIC geht, behandeln wir dieses Thema in diesem Buch nicht.

  Wenn Sie bei der Anwendung unseres Programms nicht jedesmal die "ExtrasD"-Datei einlegen wollen, können Sie auch die drei Dateien "dos.bmap", "exec.bmap" und "graphics.bmap" in die "Daten"-Schublade unserer "BASICDisk" kopieren. Entfernen Sie dann die beiden CHDIR-Befehle aus dem Programmteil "BmapSuchen:".

- Der Programmteil "GetNames:" fragt nicht mehr nach einem Namen für die ACBM-Datei ('ACBMname$'), sondern nach 'EntzILBMName$', dem Namen der entzerrten ILBM-Datei.

- Im Programmteil "EntzILBMSpeichern:" rufen wir die "Picsave"-Routine auf:

```
picsave EntzILBMName$,2,1
```

  Der Name der abzuspeichernden Datei ist 'EntzILBMName$'. Abgespeichert wird der Inhalt von Window 2, das "LoadILBM-SaveACBM" zur Darstellung der Grafik

erzeugt hat. Der Parameter 1 besagt, daß die Farbtabelle des aktuellen Bilds abgespeichert werden soll. (Näheres dazu finden Sie im Kapitel 4.8.)

- Beim Einlesen des CMAP-Chunks in "LoadILBM:" schreiben wir die gelesenen Farbwerte zusätzlich ins Feld 'Farben%'. Dort erwartet die "Picsave"-Routine nämlich die Farbtabelle.

```
Farben%(kk,0)=((red% AND 240)/16)

usw.
```

  Das Feld 'Farben%' haben wir im Programmteil "Main:" am Programmanfang mit DIM Farben%(31,2) dimensioniert.

- Im SUB-Programm "Picsave" übernehmen wir mit SHARED die Variablen 'ccrtDir%', 'ccrtStart%', 'ccrtEnd%', 'ccrtSecs&' und 'ccrtMics&'.

  In diesen Variablen stehen die Daten über eine eventuelle Farbzyklen-Animation, die wir in unserer entzerrten Datei in einem zusätzlichen Chunk (CCRT) abspeichern.

In der Zusammenfassung wurden einige Dinge erwähnt (z.B. SUB-Programme oder Variablen, die auf % oder & enden), die Sie bisher noch nicht kennen. Sie werden alle diese Dinge aber im Verlauf der nächsten Kapitel kennenlernen. Vielleicht kehren Sie dann nochmal zu diesem Zwischenspiel zurück und lesen es ein zweites Mal durch. Danach dürften die meisten Dinge klar sein.

Aber wir haben es ja schon gesagt: In erster Linie ist das "ILBMEntzerren"-Programm dazu da, Ihnen zu helfen. Sie müssen es dazu nicht unbedingt verstehen.

4.4 Die große Tauschbörse - Laden und Speichern von Bildern im Malprogramm

Jetzt ist eigentlich alles vorbereitet zur großen Tauschaktion: "Deluxe Paint" tauscht Bilder mit unserem Malprogramm, unser Malprogramm mit Graphicraft...

Nach den Vorarbeiten, die wir in den letzten beiden Kapiteln und im Zwischenspiel 5 geleistet haben, ist es auch nicht mehr schwer, die Programmteile für Laden und Speichern von IFF-Dateien in unser Malprogramm aufzunehmen. Speichern Sie also bitte das Programm aus dem letzten Kapitel, falls Sie's noch nicht getan haben, und laden Sie das Malprogramm.

In der Version des Malprogramms, die Sie auf unserer Diskette im Buch finden, sind die folgenden Teile schon eingebaut. Wenn Sie nicht selbst tippen, können Sie sich die Änderungen also sparen. Es ist aber sehr wichtig, daß Sie trotzdem die Erklärungen lesen.

Für alle anderen führen wir Schritt für Schritt die notwendigen Ergänzungen durch. Zuerst nehmen Sie bitte die folgenden beiden Zeilen im 'Projekt:'-Teil auf:

```
IF Menpunkt=4 THEN GOSUB Farbaus : GOSUB Musteraus : GOSUB Laden
IF Menpunkt=5 THEN GOSUB Farbaus : GOSUB Musteraus : GOSUB
Speichern
```

Gehen Sie dann mit dem Cursor durchs Listing, bis Sie hinter dem Programmteil 'Musteraus:' angelangt sind. Dahinter geben Sie bitte ein:

```
Laden:
  MENU 2,0,0 : MENU 1,0,0
  MENU OFF : MOUSE OFF
  GOSUB Eingabename
  WINDOW CLOSE 5
  WINDOW 2
  IF Nam$="" THEN EndeLaden
  OPEN Nam$ FOR INPUT AS 1
```

```
    Form$=INPUT$(4,1)
    Laenge=CVL(INPUT$(4,1))
    IF INPUT$(4,1)<>"ILBM" THEN BEEP : GOTO EndeLaden

Einlesen:
    IF EOF(1) THEN EndeLaden
    Chunk$=INPUT$(4,1)
    Laenge=CVL(INPUT$(4,1))
    IF INT(Laenge/2)<>(Laenge/2) THEN Laenge=Laenge-1
    IF Chunk$="BMHD" THEN BMHeader
    IF Chunk$="CMAP" THEN ColorMap
    IF Chunk$="BODY" THEN BodyMap
    Dummy$=INPUT$(Laenge,1)
  GOTO Einlesen

BMHeader:
    xd=CVI(INPUT$(2,1))
    IF xd>320 THEN BEEP : GOTO EndeLaden
    yd=CVI(INPUT$(2,1))
    IF yd>200 THEN BEEP : GOTO EndeLaden
    Dummy$=INPUT$(4,1)
    Bitebn=ASC(INPUT$(1,1))
    Dummy$=INPUT$(11,1)
    Adr=PEEKL(WINDOW(8)+4)+8
    FOR x=0 TO Bitebn-1
      Ebnadr(x)=PEEKL(Adr+4*x)
    NEXT x
  GOTO Einlesen

ColorMap:
    FOR x=0 TO (Laenge/3)-1
      r=(ASC(INPUT$(1,1)) AND 240)/16
      g=(ASC(INPUT$(1,1)) AND 240)/16
      b=(ASC(INPUT$(1,1)) AND 240)/16
      PALETTE x,r/16,g/16,b/16
      Farben%(x,0)=r : Farben%(x,1)=g : Farben%(x,2)=b
    NEXT x
    IF INT(Laenge/3)<>(Laenge/3) THEN Dummy$=INPUT$(1,1)
  GOTO Einlesen
```

```
BodyMap:
    FOR y1=0 TO 199
      FOR b=0 TO Bitebn-1
        IF b<Farben THEN
          FOR x1=0 TO 9
            POKEL Ebnadr(b)+4*x1+40*y1,CVL(INPUT$(4,1))
          NEXT x1
        ELSE
          Dummy$=INPUT$(40,1)
        END IF
      NEXT b
    NEXT y1
  GOTO Einlesen

EndeLaden:
  CLOSE 1
  MENU ON : MOUSE ON
  MENU 1,0,1 : MENU 2,0,1
RETURN

Speichern:
  MENU 2,0,0 : MENU 1,0,0
  MENU OFF : MOUSE OFF
  GOSUB Eingabename
  WINDOW CLOSE 5
  WINDOW 2
  IF Nam$="" THEN EndeSpeichern
  OPEN Nam$ FOR OUTPUT AS 1 LEN=FRE(0)-500
    PRINT #1,"FORM";
    PRINT #1,MKL$(156+8000*Farben);
    PRINT #1,"ILBM";
    PRINT #1,"BMHD";MKL$(20);
    PRINT #1,MKI$(320);MKI$(200);
    PRINT #1,MKL$(0);
    PRINT #1,CHR$(Farben);
    PRINT #1,CHR$(0);MKI$(0);MKI$(0);
    PRINT #1,CHR$(10);CHR$(11);
    PRINT #1,MKI$(320);MKI$(200);
```

```
PRINT #1,"CMAP";MKL$(96);
FOR x=0 TO 31
  PRINT #1,CHR$(Farben%(x,0)*16);
  PRINT #1,CHR$(Farben%(x,1)*16);
  PRINT #1,CHR$(Farben%(x,2)*16);
NEXT x

PRINT #1,"BODY";MKL$(8000*Farben);
Adr=PEEKL(WINDOW(8)+4)+8
FOR x=0 TO Farben-1
  Ebnadr(x)=PEEKL(Adr+4*x)
NEXT x
FOR y1=0 TO 199
  FOR b=0 TO Farben-1
    FOR x1=0 TO 9
      PRINT #1,MKL$(PEEKL(Ebnadr(b)+4*x1+40*y1));
    NEXT x1
  NEXT b
  PAdr=Ebnadr(0)+40*y1
  POKE PAdr,PEEK(PAdr) AND 63
  POKE PAdr+39,PEEK(PAdr+39) AND 252
NEXT y1

PRINT #1,"CAMG";MKL$(4);
PRINT #1,MKL$(16384);
CLOSE 1

EndeSpeichern:
  MENU ON : MOUSE ON
  MENU 1,0,1 : MENU 2,0,1
RETURN
```

Den Teil 'Laden:' kennen Sie schon größtenteils. Es ergaben sich nur ein paar kleine Änderungen, um die Leseroutine an das Malprogramm anzupassen. Im Teil 'BMHeader:', der den BMHD-Chunk liest, werden die Ausmaße der Grafik überprüft:

Ist das geladene Bild breiter als 320 oder höher als 200 Pixel, bricht die Laderoutine ab, da das Bild nicht auf den Bildschirm passen würde.

Von 'ColorMap:' werden die eingelesenen RGB-Werte ins Feld 'Farben%' gebracht, damit das Malprogramm die Farben in normaler Weise benutzen kann.

Die Schleifen im Programmteil 'BodyMap:' sind fest auf die Werte eines 320 * 200-Bilds abgestimmt, Bilder anderer Formate liest das Programm ja sowieso nicht.

Außerdem müssen wir auf die Anzahl der Bitebenen aufpassen. Wenn das eingelesene Bild mehr Bitplanes hat als der Screen, den das Malprogramm benutzt, werden die Daten aus den überzähligen Bitebenen in den 'Dummy$' gepackt und weggeworfen. Das Programm lädt nur soviele Bitebenen, wie in der Variablen 'Farben' festgelegt ist. Das hat die Folge, daß ehemals unterschiedliche Farben zwar auf dem Bildschirm gleich sein können, aber das Bild in jedem Fall geladen wird.

Mehr Neues gibt es im 'Speichern:'-Unterprogramm. Dieser Programmteil schreibt das Bild im IFF-Format auf Diskette. Der Anfang läuft nach gewohntem Muster ab: Die Pulldowns werden abgeschaltet, und ein Dateiname wird gelesen. Dann öffnet das Programm eine Datei zum Schreiben. Aber was soll denn LEN=FRE(0)-500 bedeuten? Wie Sie wissen, legt AmigaBASIC für jede Datei einen Pufferspeicher an, wo die Bytes erst gesammelt werden, bevor sie auf die Diskette kommen. Dadurch läßt sich viel Zeit sparen, denn am längsten dauert der Schreibvorgang im Diskettenlaufwerk. Normalerweise ist der Puffer 128 Bytes lang. Je mehr Platz im Puffer ist, umso schneller wird der Datenaustausch. Deshalb können Sie beim OPEN-Befehl die gewünschte Pufferlänge angeben. Das geschieht durch das Anhängen von LEN=(Puffergröße) hinter dem OPEN-Befehl. Ein Beispiel:

```
OPEN Nam$ FOR OUTPUT AS 1 LEN=1000
```

weist der Datei 1 einen Puffer von 1000 Bytes zu.

Der Haken bei der Sache ist, daß Sie nicht unbegrenzt Speicherplatz zur Verfügung haben. Der Speicherbereich, in dem das BASIC-Programm, seine Variablen und Felder sowie die Dateipuffer liegen, ist beim 512K-Amiga nur 25 KBytes groß. Erinnern Sie sich noch an die Einschaltmeldung beim Start von AmigaBASIC? Da teilt Ihnen der Amiga mit, wieviele Bytes für den BASIC-Speicherbereich zur Verfügung stehen.

Es gibt auch eine BASIC-Funktion, mit der Sie die Anzahl der freien Bytes feststellen können. Sie heißt FRE(x). Geben Sie im BASIC-Window ein:

```
? fre(0)
```

Die Zahl, die Sie als Ergebnis erhalten, sagt Ihnen die Anzahl der freien (engl.: free, daher der Funktionsname) Bytes im BASIC-Speicherbereich. Bei geladenem Malprogramm und 512 KByte RAM wird diese Zahl in der Gegend von 9000 liegen. Steht hinter FRE in Klammern die Zahl 0, gibt AmigaBASIC die freien Bytes im BASIC-Speicher an. Folgt dagegen in den Klammern hinter FRE die Zahl -1, erhalten Sie die freien Bytes im ganzen System:

```
? fre(-1)
```

Diese Zahl wird größer sein als das Ergebnis von FRE(0). Wie groß, können wir schlecht vorhersagen. Das hängt davon ab, was sich im Hintergrund tut, wieviele Windows geöffnet sind und was sich alles im Speicher befindet. Bedenken Sie, daß im Systemspeicher auch alle Grafik-Bitmaps Platz finden müssen. Wenn Sie BASIC trotzdem etwas mehr Speicher gönnen wollen: Wir werden Ihnen später zeigen, wie's geht.

Jetzt ist der Ausdruck LEN=FRE(0)-500 nicht mehr so geheimnisvoll: Als Pufferspeicher wird der verfügbare Rest des BASIC-Speichers verwendet. 500 Bytes behalten wir sicherheitshalber als Reserve für Variablen, die noch auftreten könnten, und ähnli-

ches. Wenn Sie den BASIC-Speicher restlos verbrauchen und dann auch nur noch 1 Byte benötigen, meldet AmigaBASIC einen "Out of memory"-Error.

Von dem kleinen Sicherheitsabschlag abgesehen, speichern wir so immer in der höchstmöglichen Geschwindigkeit. Nach dem Schließen der Datei wird der Pufferspeicher wieder freigegeben.

Die Datei ist geöffnet, wir können unsere Daten auf den Weg schicken.

Zuerst schreiben wir die Typ-Bezeichnung FORM in die Datei, gefolgt von der Länge der Form. Die hängt ja wiederum von der Anzahl der Bitebenen ab. Für jede Bitebene brauchen wir 8000 Bytes. 156 Bytes nehmen die Farb- und Steuerdaten in Anspruch. Der Gesamtspeicherbedarf ist also 156+8000*(Anzahl der Bitebenen).

Es folgt die Kennung ILBM (="Interleaved Bitmap", die Kennung für Bilder).

Und dann geht schon der BMHD-Chunk los. Er ist (wie üblich) 20 Bytes lang. Dort hinein kommen zuerst die Zahl 320 (Breite) und die Zahl 200 (Höhe), jeweils als 16-Bit-Zahl. Daran schließen sich vier 0-Bytes an (die Anfangskoordinaten der Grafik). Das nächste Byte beinhaltet die Anzahl der Bit-Ebenen. Und dann kommt wieder ein 0-Byte.

Die folgenden zwei Bytes kennzeichnen, ob die Grafikdaten im BODY-Chunk komprimiert gespeichert sind oder nicht. Der Wert 0 bedeutet, daß keine Kompression vorliegt. Programme, die eine Entzerr-Routine beinhalten, brauchen diese Information. Und nochmal zwei 0-Bytes. Diese Stellen werden in der gegenwärtigen Version von IFF noch nicht genutzt, sie sind für zukünftige Erweiterungen vorgesehen. Nun kommen zwei Werte: 10 und 11. Da IFF-Bilder in Zukunft einmal auf verschiedenen Computern laufen sollen, geben diese beiden Werte Auskunft darüber, in welchem Verhältnis die Breite eines Pixels zu seiner

Höhe steht. Das Seitenverhältnis kann ja bei verschiedenen Computern sehr unterschiedlich sein. Beim Amiga beträgt es 10:11, deshalb: MKI$(10);MKI$(11).

Nochmal die Zahlen 320 und 200. Diesmal geben sie nicht die Ausmaße des Bilds an, sondern die gewählte Auflösungsstufe. Das waren 20 Bytes, der BMHD-Chunk ist voll.

Kommen wir zu den Farben. Das einfachste ist, grundsätzlich 32 Farbwerte abzuspeichern, egal wieviele Farben davon im Bild wirklich verwendet werden. Wir schreiben also insgesamt 96 (32 mal 3) RGB-Werte aus dem 'Farben%'-Feld in den CMAP-Chunk. Sie erinnern sich: Mit Rücksicht auf zukünftige Hardware-Entwicklungen ermöglicht IFF die Angabe feinerer Farbabstufungen, als sie der Amiga bietet. Deshalb muß jeder Farbwert mit 16 multipliziert werden.

Jetzt kommt der Hauptteil, der BODY-Chunk: Seine Länge richtet sich nach der Anzahl der Bitebenen, pro Bitebene werden 8000 Bytes benötigt. Wir errechnen in beiden Programmteilen ('Laden:' und 'Speichern:') die Adressen der Bitebenen. Es ist ja nicht vorherzusagen, welche der beiden Routinen zuerst benutzt wird, aber wir brauchen in jedem Fall die aktuellen Daten für das Feld 'Ebnadr'.

Dieselben Schleifen, die beim Einlesen die Werte in den Speicher geschrieben haben, lesen sie jetzt mit PEEKL aus dem Speicher und schreiben sie in die Datei. Um während des Speicherns eine optische Kontrolle zu bieten, wie weit das Bild bereits gespeichert ist, haben wir die nächsten drei Programmzeilen aufgenommen.

Die Variable 'PAdr' errechnet für jeden Durchlauf die Adresse des ersten Bytes in der aktuellen Bildzeile aus der ersten Bitebene. In diesem Byte löschen wir die höchstwertigen beiden Bits (mit AND 63) und POKEn den neuen Wert zurück in die Adresse. Dasselbe machen wir mit dem (39 Bytes entfernten) letzten Byte der Bildzeile, dort werden die beiden niederwertigen Bits gelöscht (AND 252). Das Resultat: Der Windowrahmen verschwindet von oben nach unten und zwar immer auf

Höhe der Zeile, die gerade abgespeichert wurde. So erkennen Sie, wann das Programm mit dem Abspeichern fertig ist. Nach dem Speichern wird der Rahmen beim ersten Klick mit der Menütaste der Maus wieder aufgebaut.

Zu guter Letzt speichern wir noch einen Chunk, den unser eigenes Programm zwar gar nicht liest, der aber von Graphicraft und manchen anderen Profis benötigt wird: Der CAMG-Chunk (steht schlicht und einfach für: Commodore AMiGa) ist nur vier Bytes groß und beinhaltet eine 32-Bit-Zahl, die für Intuition Informationen über die Grafik-Betriebsart liefert.

Ist auch der letzte Chunk gut auf den Weg zur Diskette gebracht, können wir die Datei schließen.

Das war's. Nun ist das Malprogramm IFF-kompatibel. Sie können Bilder laden und abspeichern und so die Vorteile anderer Grafikprogramme für Ihre BASIC-Bilder nutzen. Professionelle Malprogramme bieten Ihnen die Möglichkeit, Bildausschnitte zu kopieren, zu vergrößern oder Hardcopies von Ihren Bildern zu drucken. Und noch manches mehr. Dafür hat unser BASIC-Malprogramm den Füllmuster-Editor, der Ihnen auch für Graphicraft- oder Deluxe Paint-Bilder gute Dienste leisten wird.

Sie sehen: Amiga hält mehr von Miteinander als Gegeneinander. Getreu diesem Motto wollen wir jetzt auch das Videotitel-Programm in den illustren Kreis IFF-kompatibler Programme aufnehmen.

Zwischenspiel 6: Ein Programm speckt ab - die Änderungen am Videotitel-Programm

Wie finden Sie das: Sie nehmen ein Bild, das mit Graphicraft erstellt wurde, fügen im Malprogramm ein paar Muster ein und laden dieses Bild als Hintergrund in das Videotitel-Programm. Davor läuft der Text ab, und das bewegte Grafikobjekt schwebt

durch die Gegend. Das wäre doch ganz reizvoll, oder? Um das zu erreichen, müßten Sie lediglich das Videotitel-Programm um eine IFF-Leseroutine erweitern.

Es gibt nur ein Problem, an das Sie jetzt vielleicht nicht denken: die Auflösungsstufen. Wie Sie wissen, sind alle Graphicraft-Bilder und die weitaus überwiegende Anzahl an IFF-Bildern anderer Malprogramme 320 * 200 Pixels groß. Unser Malprogramm arbeitet ebenfalls nur in dieser Darstellung. Sie erinnern sich aber vielleicht, daß das Videotitel-Programm in der BASIC-Standardauflösung (640 * 200 Pixels) läuft. Die IFF-Bilder wären als Hintergrund einfach zu klein. Und sie hätten möglicherweise zuviele Farben (32 statt 16). Was tun?

Die einfachste und sinnvollste Lösung ist, das Videotitel-Programm an die niedrige Auflösung anzupassen. Viel muß da sowieso nicht geändert werden, in erster Linie geht es um die Textdarstellung auf dem Bildschirm. Die Folgen solcher Änderungen, nämlich die größeren Buchstaben, sind eigentlich eher von Vorteil: Breitere Schrift ist auf einer Videoaufzeichnung einfacher zu lesen als die 80 Zeichen pro Zeile der Workbench.

In diesem Zwischenspiel wollen wir die notwendigen Umstellungen am Videotitel-Programm gemeinsam durchführen. Den Eintipp-Abstinenzlern sei gleich verraten, daß Sie die dabei entstehende Version auch wieder auf unserer Diskette im Buch finden. Sie heißt "Videotitel.IFF" und befindet sich sowohl in der "Daten"- als auch in der "Videotitel"-Schublade.

Wenn Sie jedoch selbst tippen wollen, dann laden Sie bitte Ihr "Videotitel"-Programm, das sich in der gleichnamigen Schublade auf der "BASICDisk" befinden müßte.

Wenn Sie während der folgenden Arbeiten das Programm abspeichern, verwenden Sie bitte einen neuen Namen (wie z.B. "Videotitel.IFF"), damit Ihnen beide Versionen des Videotitel-Programms erhalten bleiben.

Wir zeigen Ihnen alle Stellen, an denen etwas geändert werden muß. Beginnen wir am Programmanfang. Im Teil 'Vorbereitungen:' löschen Sie bitte die IF...THEN-Bedingung vor der SCREEN-Anweisung. Während wir in der alten Version nur dann einen neuen Screen eröffneten, wenn mehr als 2 Bitebenen gefragt waren, brauchen wir in der neuen Version in jedem Fall einen. Die Angaben beim SCREEN-Befehl werden auf 320 * 200 Pixels umgeschrieben. Nach allen Änderungen sieht die fünfte Zeile in 'Vorbereitungen:' dann so aus:

Vorher:

```
IF Farben>2 THEN SCREEN 2,640,200,Farben,2 : WINDOW 2,"Videotitel",,28,2
```

Nachher:

```
SCREEN 2,320,200,Farben,1 : WINDOW 2,"Videotitel",,28,2
```

Eine Zeile darunter stehen eine Reihe DIM-Anweisungen. Das 'Farbfeld' sollte auf die höchstmögliche Farbenzahl vorbereitet werden. Ändern Sie bitte die Angabe Farbfeld(d,3) in Farbfeld(31,3):

Vorher:

```
DIM Text$(d),Farbfeld(d,3),Beweg(d),Geschw(d)
```

Nachher:

```
DIM Text$(d),Farbfeld(31,3),Beweg(d),Geschw(d)
```

Nach dieser unscheinbaren Änderung sind wir im Teil 'Vorbereitungen:' auch schon fertig mit unseren Bauarbeiten. Kommen wir zum 'Anfang:'

Vorher:

```
PRINT "Videotitel-Programm ";
```

Nachher:

```
PRINT "Videotitel-Programm "
```

Der Strichpunkt hintendran ist weg, das ist alles. So erzeugen wir einen Zeilenvorschub zwischen "Videotitel-Programm" und Ihrem Namen. Anderenfalls verschwindet ein Teil der Überschrift nämlich hinter dem rechten Bildschirmrand.

Jetzt kommt ein längeres, änderungsfreies Stück. Erst wieder zu Beginn der Teils 'ObjEinlesen:' gibt es eine Kleinigkeit zu tun:

Vorher:

```
PRINT "Bitte geben Sie den Namen des Objekts ein."
```

Nachher:

```
PRINT "Bitte geben Sie" : PRINT "den Namen des Objekts ein."
```

Auch hier ist unser einziges Bestreben, aus dem zu langen Einzeiler einen passenden Zweizeiler zu machen. Dasselbe fällt auch am Beginn von 'Bewegen:' an, sogar gleich doppelt:

Vorher:

```
PRINT "Legen Sie bitte die Ausgangsposition fest."
PRINT "Bewegen Sie das Objekt mit den Cursortasten."
```

Nachher:

```
PRINT "Legen Sie bitte die" : PRINT "Ausgangsposition fest."
PRINT "Bewegen Sie das Objekt" : PRINT "mit den Cursortasten."
```

Auch ein Stück weiter unten, im Programmteil 'Zieldef:' gilt es wieder, eine Zeile zu verhackstücken:

Vorher:

```
PRINT "Bitte bewegen Sie das Objekt zum"Ziel". Zielpunkt."
```

Nachher:

```
PRINT "Bitte bewegen Sie das Objekt" : PRINT "zum"Ziel". Ziel-
punkt."
```

Bevor Sie das Gefühl haben, wir würden Sie völlig unterfordern, kommt jetzt mal ein etwas größerer Teil. Die Bildschirmdarstellung in 'Farben:' muß auf den verringerten Platz und die erhöhte Farbanzahl vorbereitet werden:

Vorher:

```
Farben:
  FOR x=0 TO Maxfarbe
    COLOR -(x=0),x
    LOCATE 5,(x*4) + 1
    PRINT x;CHR$(32);CHR$(32)
  NEXT x
```

Nachher:

```
Farben:
  FOR x=0 TO Maxfarbe
    IF (x/8)=INT(8) THEN PRINT
    COLOR -(x=0),x
    PRINT x;
    IF x<10 THEN PRINT CHR$(32);
  NEXT x
```

Die Sache ist eigentlich ganz einfach. IF (x/8)=INT(8)... bedeutet: Wenn x ein ganzes Vielfaches von 8 ist (also: 8, 16 oder 24), soll ein Zeilenvorschub erfolgen, ansonsten werden die Farbfelder nebeneinander gedruckt. Und IF x<10 THEN ... bewirkt, daß die Felder der Zahlen 0 bis 9 um eine Leerstelle erweitert werden, weil die gedruckten Zahlen ja nur einstellig sind. Ab 10

sind die Zahlen zweistellig. (Logisch! Wer's nichts glaubt, darf alle Zahlen von 0 bis 99 auf einen Zettel schreiben und dann die Stellen nachzählen...)

Im nächsten Teil 'Farbaenderung:' gibt es wieder einen Satz, der sich auseinandergelebt hat:

Vorher:

```
PRINT "Welche Farbe soll geändert werden?"
```

Nachher:

```
PRINT "Welche Farbe" : ?"soll geändert werden?"
```

Die Änderungen, die wir im Teil 'RGBRegler:' durchzuführen haben, dienen alle wieder der Anpassung auf den kleineren Textbildschirm. Die Regler für R, G und B und die zugehörigen Erklärungen müssen auf zwei Zeilen verteilt werden:

Vorher:

```
LOCATE 10,1 : PRINT "Rot:   <7>=- <8>=+ ";Fuell$
LOCATE 10,20+r : PRINT CHR$(124);
LOCATE 11,1 : PRINT "Grün: <4>=- <5>=+ ";Fuell$
LOCATE 11,20+r : PRINT CHR$(124);
LOCATE 12,1 : PRINT "Blau:  <1>=- <2>=+ ";Fuell$
LOCATE 12,20+r : PRINT CHR$(124);
LOCATE 13,1 : PRINT "       <0>=Farbe o.k."
```

Nachher:

```
LOCATE 11,1 : PRINT "Rot:   <7>=- <8>=+ " : PRINT Fuell$
LOCATE 12,1+r : PRINT CHR$(124);
LOCATE 13,1 : PRINT "Grün: <4>=- <5>=+ " : PRINT Fuell$
LOCATE 14,1+r : PRINT CHR$(124);
LOCATE 15,1 : PRINT "Blau:  <1>=- <2>=+ " : PRINT Fuell$
LOCATE 16,1+r : PRINT CHR$(124);
LOCATE 17,1 : PRINT "       <0>=Farbe o.k."
```

Na, haben Sie die 13 kleinen Unterschiede gefunden? Es liegt wirklich nicht in unserer Absicht, Ihnen irgendwelche Rätsel aufzugeben, aber in diesem Fall sollten Sie wirklich genau vergleichen:

Erstens rutschen die Werte in den LOCATE-Zeilen um 1 nach unten und werden nun durchgehend numeriert. Zweitens werden die LOCATE-Spalten von 20+r auf 1+r geändert. Drittens und letztens wird der 'Fuell$' am Ende der Zeilen mit einem eigenen PRINT-Befehl versehen.

Die nächste Etappe ist wieder ein Stück entfernt. In den Teilen 'Farbeingabe:' und 'Schleife3:' müssen wir mit der LOCATE-Zeile nach unten ausweichen.

Vorher:

```
LOCATE 14,1
```

Nachher:

```
LOCATE 19,1
```

Achtung: Die letzte Korrektur muß an zwei kurz hintereinander folgenden Stellen ausgeführt werden!

Beim 'Countdown:' gibt es wieder eine kleine Verschiebung, diesmal innerhalb der Zeile auf eine kleinere Spaltennummer.

Vorher:

```
LOCATE 10,38 : PRINT c
```

Nachher:

```
LOCATE 10,18 : PRINT c
```

Die Zeilenbreite, die vom WIDTH-Befehl festgelegt wird, muß im Teil 'WiedergStart:' verringert werden: Von 60 auf 32.

Vorher:

```
WITDH 80
```

Nachher:

```
WIDTH 40
```

Auch die Berechungsformeln für die Textzentrierung müssen wir auf die neue Zeilenbreite anpassen. Ändern Sie auch hier den Wert 78 in 39 und den Wert 80 in 40:

Vorher:

```
Text$=LEFT$(Text$(x),78)
h=INT((80-LEN(Text$))-2)+2
```

Nachher:

```
Text$=LEFT$(Text$(x),39)
h=INT((40-LEN(Text$))-2)+2
```

Tja. Das war unsere kleine Anleitung: "Wie bastle ich mir ein IFF-kompatibles Videoprogramm". Durch unsere Arbeiten läuft das Programm jetzt im 320 * 200-Pixel-Modus. Probieren Sie die einzelnen Funktionen bitte aus, damit Sie den neuen Bildschirmaufbau kennenlernen und eventuelle Eintippfehler finden.

Vergessen Sie nicht, das Programm unter neuem Namen abzuspeichern, falls Sie die Änderungen selbst vorgenommen haben. Wenn alles erledigt ist, dürfen wir Sie ins nächste Kapitel bitten, wo der IFF-Teil dazukommt.

4.5 Action! - Einlesen von Bildern im Videotitel-Programm

Auch das Videotitel-Programm braucht also eine IFF-Leseroutine. Wir wollen Ihnen ersparen, das weitgehend gleiche Programm zum dritten Mal abzutippen. Deshalb benutzen wir ein paar Tricks, die wir bei der Arbeit mit Peripheriegeräten ken-

nengelernt haben. Das Videotitel-Programm haben Sie ja sicher schon längst abgespeichert, bitte laden Sie jetzt das Malprogramm. Bedenken Sie, daß Sie dafür vorher wahrscheinlich mit CHDIR das aktuelle Inhaltsverzeichnis wechseln müssen.

Das alles betrifft natürlich wieder nur diejenigen, die selbst an dem Programm arbeiten wollen. Wer sich lieber der beigelegten Diskette im Buch bedient, findet wie schon im letzten Zwischenspiel erwähnt, die IFF-fähige Version des Videotitel-Programms in der "Daten" oder in der "Videotitel"-Schublade unter dem Namen "Videotitel.IFF". Nach wie vor gilt auch, daß trotzdem alle die Erklärungen lesen sollten, um zu verstehen, was in den neuen Programmteilen passiert.

Im Malprogramm haben wir eine Routine, die Bilder mit 320 * 200 Pixels einliest. Mit wenigen kleinen Änderungen können alle Selbst-Tipper die Routine ins Videotitel-Programm übernehmen. Die folgende Zeile, im BASIC-Window eingegeben, speichert den Einleseteil auf der RAM-Disk als ASCII-Datei ab:

```
list Laden-EndeLaden,"ram:iff.lesen"
```

Alle Zeilen zwischen den Labels 'Laden:' und 'EndeLaden:' werden als Datei "iff.lesen" abgespeichert. Löschen Sie danach das Malprogramm mit NEW und laden Sie wieder die neue Version des Videotitel-Programms.

Die IFF-Leseroutine hängen wir jetzt einfach hinten an:

```
merge "ram:iff.lesen"
```

Und schon haben wir das Rohmaterial komplett. Jetzt gibt es nur noch Feinarbeiten: Die neue Routine muß in das Programm eingebunden werden. Zuerst gönnen wir dem Bilder-Einlesen einen eigenen Menüpunkt. Nehmen Sie ihn im Teil 'Auswahl:' unterhalb von Punkt 5 auf:

```
PRINT "6 Hintergrundbild festlegen"
```

Was wir so großartig auf dem Bildschirm ankündigen, muß der Benutzer natürlich auch auswählen können. Deshalb führen Sie bitte im Programmteil 'Abfrage:' folgende Änderungen durch:

Vorher:

```
IF a$<"1" OR a$>"5" THEN BEEP : GOTO Abfrage
```

Nachher:

```
IF a$<"1" OR a$>"6" THEN BEEP : GOTO Abfrage
```

Die Nummer der höchsten erlaubten Eingabe ist 6, nicht mehr 5. Wenige Zeilen später, in die Serie von IF...THEN-Abfragen, fügen wir den Aufruf für den neuen Programmteil ein:

```
IF a$="6" THEN BildVorbereiten
```

Da eine neue Zeile dazugekommen ist, müssen Sie die Eingabezeile um 1 nach unten versetzen: Ändern Sie im Teil 'Abfrage:' die Zeilenangabe der LOCATE-Anweisung:

Vorher:

```
LOCATE 10,1
```

Nachher:

```
LOCATE 11,1
```

Den neuen Teil 'BildVorbereiten:' schreiben wir zwischen den Teilen 'GeschwBerechnen:' und 'Laden:' ins Programm:

```
BildVorbereiten:
  CLS
  PRINT "Soll als Hintergrund eine"
  PRINT "Grafik geladen werden? (J/N)"
```

```
Schleife5:
  LOCATE 2,29 : INPUT Antw$
  IF UCASE$(Antw$)="N" THEN IFF=0 : CLS : GOTO Anfang
  IF UCASE$(Antw$)="J" THEN IFF=1 : GOTO EingabeName
GOTO Schleife5

EingabeName:
  PRINT
  PRINT "Bitte geben Sie den Namen ein:"
  INPUT Nam$
  PRINT
  PRINT "Soll die Farbtabelle von"
  PRINT Nam$
  PRINT "verwendet werden? (J/N)";
Schleife6:
  LOCATE 9,24 : INPUT Antw$
  IF UCASE$(Antw$)="N" THEN IFFTab=0 : CLS : GOTO Anfang
  IF UCASE$(Antw$)="J" THEN IFFTab=1 : CLS : GOTO Anfang
GOTO Schleife6
```

Im Programmteil 'BildVorbereiten:' fragt der Amiga zuerst, ob bei der Titelwiedergabe eine Hintergrundgrafik geladen werden soll oder nicht. So können Sie ausprobieren, wie sich die Grafik macht, und sie bei Nichtgefallen auch wieder abschalten. Wir vermuten aber stark, daß Sie auf die tollen Möglichkeiten der Hintergrundgestaltung gar nicht mehr verzichten wollen, wenn Sie die ersten Ergebnisse gesehen haben. Sie können die Frage mit "J" oder "N" beantworten. Alle anderen Eingaben führen zurück zur Eingabeschleife. Die Variable 'IFF' gibt später im Programm Auskunft darüber, ob eine IFF-Grafik geladen werden soll (IFF=1) oder nicht (IFF=0).

Der Programmteil 'EingabeName:' fragt nach dem Namen der Grafik, der in 'Nam$' gespeichert wird, und stellt die Frage, ob die Farbtabelle der gewünschten Grafik (die Farben aus dem CMAP-Chunk) verwendet werden soll oder nicht.

Dahinter steckt folgendes: Der Text und die Grafik müssen mit denselben Farben auskommen. Vielleicht legen Sie ja mehr Wert auf die Text- und Hintergrundfarben als auf die farblich rich-

tige Darstellung der Grafik. In diesem Fall können Sie auf die letzte Frage mit "N" antworten. Dann verwendet das Programm bei der Wiedergabe die Farbtabelle, die Sie im Videotitelprogrmm selbst festgelegt haben. Dadurch wird die Hintergrundgrafik natürlich in völlig falschen Farben dargestellt. In der Praxis müssen Sie sich wohl für Kompromisse entscheiden. Die Variable 'IFFTab' hilft dem Programm, sich Ihre Entscheidung zu merken.

Jetzt müssen wir uns noch um das 'Laden:'-Unterprogramm kümmern. Wir wollen Ihnen bei dieser Gelegenheit eine neue Möglichkeit zeigen, Unterprogramme an bestehende Programme anzuhängen. Bei der bisherigen Methode, wo wir die neuen Programmteile einfach mit GOTO oder GOSUB ausführen lassen, kann nämlich ein Problem auftreten:

In unserem IFF-Ladeprogramm verwenden wir ziemlich viele Variablen. Dieselben Variablen könnten aber auch im Hauptprogramm benötigt werden. Zwar ist die Wahrscheinlichkeit gering, daß in einem Ihrer Programme Variablennamen wie 'Form$', 'Laenge' oder 'Chunk$' vorkommen, aber wie steht's mit 'x' und 'y'? Wo doch 'x' und 'y' sowieso so eine Art Dick und Doof der Mathematik sind. (Das ist übrigens unser einziger, aber sehr beliebter Standardwitz in jedem Buch...) Stellen Sie sich vor, in Ihrem Programm hat 'x' den Wert 3000. Es handelt sich beispielsweise um ein Spielprogramm, und 'x' ist die Anzahl der Punkte, die Sie schon erreicht haben. Jetzt laden Sie eine neue Hintergrundgrafik, und das Programm kehrt aus dem Lade-Teil zurück. Die Variable 'x' wird als Zählvariable im 'ColorMap:'-Teil benutzt und hat nach der Rückkehr den Wert 31. Sehr schnell hätten Sie eine ganze Menge Punkte verloren. Das mag noch kein Beinbruch sein, aber was passiert, wenn 'x' Teil einer wichtigen Berechnung ist? Sie sehen: Die Verwendung gleicher Variablen in Haupt- und Unterprogramm kann sehr unangenehme Folgen haben. Eine Lösung wäre, in Ihren Unterprogrammen nur exotische Variablennamen zu verwenden, aber eine absolute Sicherheit bietet selbst das nicht.

Abhilfe schafft der Befehl SUB...STATIC. Mit ihm wird ein spezieller Typ von Unterprogrammen eingeleitet. Wir wollen diesen Typ ab jetzt SUB-Programme nennen. (Obwohl "SUB-Programm" und "Unterprogramm" streng genommen dasselbe bedeutet.) SUB-Programme unterscheiden sich von den gewohnten Unterprogrammen durch ihre Variablenverwaltung: In SUB-Programmen werden sogenannte lokale Variablen verwendet. Das heißt, alle Variablen innerhalb dieses Programmteils haben eigene Werte, die mit den Variablen des Hauptprogramms nichts zu tun haben. Wenn es im SUB-Programm eine Variable 'x' gibt und im Hauptprogramm eine gleichnamige Variable, beeinflussen sich die Werte überhaupt nicht.

Sie können im Hauptprogramm den Wert von 'x' ändern, der Wert 'x' im SUB-Programm bleibt unverändert und umgekehrt. Möglich wird das dadurch, daß das SUB-Programm einen eigenen Speicherbereich für seine Variablen anlegt. Wenn AmigaBASIC aus dem SUB-Teil ins Hauptprogramm springt und später wieder ins SUB-Programm zurückkehrt, wird es dort dieselben Variableninhalte wie beim letzten Mal vorfinden. Auf diese Eigenschaft weist das Wort STATIC hin, das zum Befehl SUB dazugehört ("Static" heißt "statisch, unverändert").

Nun ein paar Worte zur Programmierung von SUB-Programmen. Zwischen SUB und STATIC steht der Name der Routine. Zum Beispiel:

```
SUB Umrechnen STATIC
```

Diese Zeile gehört nicht zum Videotitel-Programm, sie ist nur als Beispiel gedacht. Am Ende der Routine muß der Befehl END SUB stehen. SUB-Programme können an jeder beliebigen Stelle im Programm stehen. Durch die Umklammerung mit SUB...STATIC und END SUB erkennt AmigaBASIC, daß der Teil nicht zum Hauptprogramm gehört. Die Routine wird selbst dann nicht abgearbeitet, wenn sie mitten im Programm steht. Im folgenden Beispiel werden nur die erste und die letzte Programmzeile ausgeführt:

```
x=100

SUB Umrechnen STATIC
  x=40
  x=x*100+3
  PRINT "Hallo!"
END SUB

PRINT x
```

Auch diese Zeilen sollten Sie nicht abtippen, wenn Sie gerade am Videotitel-Programm arbeiten. Die gehören nämlich noch nicht dazu.

Bei der Programmierung von SUB-Programmen gibt es einige Regeln zu beachten: Sie dürfen zum Beispiel innerhalb einer SUB/END SUB-Klammer keine weitere SUB-Routine definieren. Das Folgende ist also verboten:

```
SUB Test STATIC
  SUB Test2 STATIC
    PRINT "Test2"
  END SUB
  PRINT "Test"
END SUB
```

Die Zeilen oben erzeugen einen "Tried to declare SUB within a SUB"-Error. Also auf Deutsch: "Sie haben versucht, ein SUB-Programm innerhalb eines SUB-Programms zu definieren. Das geht aber nicht. Mit freundlichen Grüßen, Amiga Fehlerabteilung."

Wenn zwei gleichnamige SUB-Programme vorkommen, gibt es einen "SUB already defined"-Error ("SUB-Programm bereits definiert.").

Wenn Sie STATIC hinter SUB vergessen, meldet der Amiga: "Missing STATIC in SUB statement". Also: "STATIC fehlt im SUB-Befehl."

Und auch das END SUB ist unbedingt nötig. Wenn Sie's vergessen, erinnert Sie AmigaBASIC mit: "SUB without END SUB" - SUB ohne END SUB.

Wenn aber alles stimmt mit so einer SUB-Routine, wie ruft man sie dann auf? Dazu gibt es den Befehl CALL. Zum Beispiel:

```
CALL Umrechnen
```

Wenn es keine Verwechslungen geben kann, dürfen Sie das CALL sogar weglassen und einfach den Namen der SUB-Routine angeben:

```
Umrechnen
```

Sie werden sicher fragen: "Wann können denn Verwechslungen auftreten?" Wir wollen Ihnen ein paar Beispiele zeigen, die alle wieder nicht zum Videotitel-Programm gehören. Aber keine Sorge, lange dauert's nicht mehr, dann wenden wir unsere neuen Kenntnisse zum Bilder-Einlesen an.

```
FOR x=1 TO 10 : Umrechnen : NEXT x
```

Diese Zeile ist eindeutig. Innerhalb der FOR...NEXT-Schleife wird das SUB-Programm 'Umrechnen' aufgerufen. Nicht eindeutig ist der nächste Fall:

```
Umrechnen: PRINT x
```

Ist 'Umrechnen:' ein SUB-Aufruf, gefolgt von einem einem Doppelpunkt, oder ein Label, das zum PRINT-Befehl gehört? In diesem Fall entscheidet sich AmigaBASIC für das Label. Zur Klarstellung ist ein CALL nötig. Also:

```
CALL Umrechnen: PRINT x
```

Gleich ein ähnlicher Fall:

```
IF x=10 THEN Umrechnen
```

Ist 'Umrechnen' ein Label, zu dem verzweigt werden soll, oder ein SUB-Programm? Auch hier ist CALL zwingend, also:

```
IF x=10 THEN CALL Umrechnen
```

Jetzt sind die wichtigsten Fragen geklärt, also machen wir nun endlich am Videotitel-Programm weiter: Aus der 'Laden:'-Routine wollen wir ein SUB-Programm machen. Bei dieser Gelegenheit führen wir auch gleich die anderen nötigen Änderungen an der IFF-Leseroutine aus.

Machen Sie zunächst aus dem Label 'Laden:' die Zeile:

```
SUB Laden STATIC
```

Und das gehört jetzt endlich mal wieder zum Videotitel-Programm! Also vergessen Sie nicht, es einzugeben. Wenn Sie überhaupt mittippen. Löschen Sie dann bitte den Programmteil bis zur Zeile

```
IF Nam$="" THEN EndeLaden
```

Also die ersten 4 Zeilen der SUB-Routine 'Laden'. An ihre Stelle muß etwas Neues. Vielleicht haben Sie sich darüber auch schon Gedanken gemacht: Wenn die SUB-Routine ihre eigenen Variablen verwendet und die Variablen des Hauptprogramms überhaupt nicht zur Kenntnis nimmt, was soll man dann tun, wenn man einzelne Werte aus dem Hauptprogramm unbedingt braucht?

Wir brauchen zum Beispiel zum Laden von IFF-Bildern dringend die Anzahl der Bitebenen aus der Variablen 'Farben', außerdem das Feld 'Farbfeld', um dort die Farbwerte abzuspeichern, die Variable 'IFFTab', damit wir wissen, wie es die Leseroutine mit der Farbtabelle halten soll, und, last but not least, den Dateinamen 'Nam$'. Was tun?

Glücklicherweise haben die Entwickler von AmigaBASIC genausoweit gedacht und für uns einen Befehl vorgesehen: SHARED. Fügen Sie bitte direkt unter der SUB...STATIC-Anweisung die nächste Zeile ein:

```
SHARED Farben,Farbfeld(),IFFTab,Nam$
```

Das englische Wort "to share" heißt "aufteilen, sich beteiligen". Das Hauptprogramm und das SUB-Programm teilen sich die angegebenen Variablen. Das heißt, sie benutzen sie gemeinsam. Sie können beim SHARED-Befehl die Variablen und Felder angeben, die in beiden Programmteilen denselben Wert haben. Wie Sie oben sehen, müssen Sie hinter dem Namen eines Datenfelds ein leeres Klammernpaar angeben. Damit unterscheiden Sie Feldnamen von Variablennamen. Und schon sind alle Probleme gelöst.

Die nächsten Änderungen gibt es erst wieder im 'ColorMap:'-Teil. So sieht der Teil vorher aus:

```
ColorMap:
  FOR x=0 TO (Laenge/3)-1
    r=(ASC(INPUT$(1,1)) AND 240)/16
    g=(ASC(INPUT$(1,1)) AND 240)/16
    b=(ASC(INPUT$(1,1)) AND 240)/16
    PALETTE x,r/16,g/16,b/16
    Farben%(x,0)=r : Farben%(x,1)=g : Farben%(x,2)=b
  NEXT x
  IF INT(Laenge/3)<>(Laenge/3) THEN Dummy$=INPUT$(1,1)
GOTO Einlesen
```

Und so nachher:

```
ColorMap:
  FOR x=0 TO (Laenge/3)-1
    r=(ASC(INPUT$(1,1)) AND 240)/16
    g=(ASC(INPUT$(1,1)) AND 240)/16
    b=(ASC(INPUT$(1,1)) AND 240)/16
    IF IFFTab=1 THEN
      PALETTE x,r/16,g/16,b/16
```

```
        Farbfeld(x,1)=r : Farbfeld(x,2)=g : Farbfeld(x,3)=b
      END IF
    NEXT x
    IF INT(Laenge/3)<>(Laenge/3) THEN Dummy$=INPUT$(1,1)
  GOTO Einlesen
```

Es dreht sich also nur um zwei Dinge: Eine IF...THEN-Bedingung muß eingefügt werden: Nur wenn 'IFFTab' den Wert 1 hat, sollen die gelesenen Farben wirklich übernommen werden. Und das Feld 'Farben%' aus dem Malprogramm machen wir im Videotitel-Programm zum Feld 'Farbfeld'. Die zweite Dimension des Felds wird außerdem um eins nach oben verschoben. Der Rot-Wert der Farbe 'x' steht in 'Farben%(x,0)', aber in 'Farbfeld(x,1)'. Dasselbe gilt für Grün (von 1 auf 2) und für Blau (von 2 auf 3). Das war hier auch schon alles.

Die letzte Änderung: Der Teil 'EndeLaden:' wird deutlich verkürzt, er besteht jetzt nur noch aus zwei Zeilen:

```
EndeLaden:
CLOSE 1
END SUB
```

Wir haben es fast geschafft. Die IFF-Leseroutine ist an das Videotitel-Programm angepaßt und umgekehrt. Wir müssen bloß noch einen Aufruf im Hauptprogramm unterbringen. Bewegen Sie dazu den Cursor zum Programmteil 'WiedergStart:' und fügen Sie den CALL-Befehl ein:

```
WiedergStart:
  WIDTH 32
  COLOR Textfa,Hintgr : CLS
  COLOR Textfa,Texthi
  IF IFF=1 THEN CALL Laden
  FOR x=1 TO Zeilen
    .
    .
    .
```

Damit haben Sie das Einlesen von Hintergrundbildern fest eingebaut. Vergessen Sie bitte nicht, das Programm gleich auf Diskette abzuspeichern.

Die Bedienung des neuen Programmteils ist nicht schwer: Wählen Sie den Menüpunkt 6 in der Auswahl, wenn Sie ein Hintergrundbild verwenden oder das Laden wieder abschalten wollen. Dort können Sie den Dateinamen eingeben und festlegen, ob die gerade eingestellten Farben oder die Farben aus der IFF-Datei verwendet werden sollen. Nach dem Countdown und vor der Darstellung von Text und Bewegung wird dann bei der Wiedergabe das Bild geladen. Noch ein Hinweis: Beim Laden wird ja wieder der Windowrahmen gelöscht. Für Videotitel ist das gar nicht so schlecht. Sollten Sie aber doch lieber im vertrauten Rahmen bleiben wollen, genügt ein Klick mit der Menütaste der Maus.

Übrigens, zur Maus noch ein Tip: Wissen Sie, wie Sie ein zweites, voll bewegliches Grafikobjekt für Ihren Videotitel verwenden können? Definieren Sie einfach mit Preferences den Mauscursor um und steuern Sie das zweite Objekt während der Aufnahme mit der Maus.

Jetzt wünschen wir Ihnen viel Spaß und hoffentlich gelungene Videotitel. Ein letztes Bonbon für die Freunde des Videotitel-Programms haben wir noch. Was das ist, sehen Sie im nächsten Kapitel. Wenn Sie jetzt ein bißchen neugierig sind: Genau das haben wir beabsichtigt.

4.6 Und noch 'ne Idee - Laden und Speichern von Titelsequenzen

Wenn Ihnen ein Titel, den Sie mit unserem Programm erzeugt haben, besonders gut gefällt, finden Sie es wahrscheinlich ärgerlich, daß Sie die einzelnen Parameter immer wieder eingeben müssen. Warum nicht einfach alle Daten auf Diskette abspeichern und bei Bedarf wieder laden?

Genau dafür stellen wir Ihnen jetzt die nötigen Erweiterungen vor. Sie sehen: Auch aus dem Videotitel-Programm wird mehr und mehr ein komfortables und leistungsfähiges Programm.

Die folgenden Teile geben Sie bitte am bisherigen Ende des Videotitel-Programms ein. Oder sie halten sich einfach wieder an die Version "Videotitel.IFF" auf der beiliegenden Diskette, wo auch diese Ergänzungen bereits berücksichtigt sind.

```
TitelSpeichern:
  CLS : PRINT "Bitte Dateinamen eingeben:"
  INPUT DatName$
  OPEN DatName$ FOR OUTPUT AS 1
    PRINT #1,Zeilen          : REM Anzahl Textzeilen
    FOR x=1 TO Zeilen
      WRITE #1,Text$(x)
    NEXT x

    PRINT #1,Eingel          ' Objekt eingelesen?
    WRITE #1,Objname$        ' Dateiname

    PRINT #1,Beweg(0)        ' Anzahl Bewegungsdaten
    FOR x=1 TO Beweg(0)
      PRINT #1,Beweg(x)
    NEXT x

    PRINT #1,Farben          ' Anzahl Bitebenen
    FOR x=0 TO 31            ' 32 Farben in IFF-Speicherung
      PRINT #1,CHR$(Farbfeld(x,1)*16);
      PRINT #1,CHR$(Farbfeld(x,2)*16);
      PRINT #1,CHR$(Farbfeld(x,3)*16);
    NEXT x
    PRINT #1,Hintgr          ' Schriftfarben etc.
    PRINT #1,Textfa
    PRINT #1,Texthi

    PRINT #1,IFF             ' Hintergrundbild?
    PRINT #1,IFFTab          ' Farben übernehmen?
    WRITE #1,Nam$            ' Dateiname
```

```
  CLOSE 1
  CLS
GOTO Anfang

TitelLesen:
  CLS : PRINT "Bitte Dateinamen eingeben:"
  INPUT DatName$
  OPEN DatName$ FOR INPUT AS 1
    INPUT #1,Zeilen
    FOR x=1 TO Zeilen
      INPUT #1,Text$(x)
    NEXT x

    INPUT #1,Eingel
    INPUT #1,Objname$

    IF Eingel=1 THEN
      OPEN Objname$ FOR INPUT AS 2
        OBJECT.SHAPE 1,INPUT$(LOF(2),2)
      CLOSE 2
    END IF

    INPUT #1,Beweg(0)
    FOR x=1 TO Beweg(0)
      INPUT #1,Beweg(x)
    NEXT x

    INPUT #1,Farben1
    IF Farben1<=Farben THEN Farben=Farben1
    CCMaxfarbe=(2^Farben)-1
    FOR x=0 TO 31
      r=(ASC(INPUT$(1,1) AND 240)/16
      g=(ASC(INPUT$(1,1) AND 240)/16
      b=(ASC(INPUT$(1,1) AND 240)/16
      PALETTE x,r/16,g/16,b/16
      Farbfeld(x,1)=r : Farbfeld(x,2)=g : Farbfeld(x,3)=b
    NEXT x
    INPUT #1,Hintgr
```

```
    INPUT #1,Textfa
    INPUT #1,Texthi

    INPUT #1,IFF
    INPUT #1,IFFTab
    INPUT #1,Nam$
  CLOSE 1
  CLS
GOTO Anfang
```

Die einzelnen Funktionen beim Schreiben und Lesen kennen Sie mittlerweile so gut, daß wir uns große Erklärungen sicher sparen können. Einige Hinweise auf Punkte, wo vielleicht doch Unklarheiten auftreten:

Wir verwenden für die Speicherung der Farben dieselbe Schreibweise wie beim IFF-Format, weil sie die platzsparendste ist. Die erzeugte Datei ist natürlich trotzdem nicht IFF-kompatibel.

Der WRITE-Befehl verhindert Probleme, die durch Kommas in Strings entstehen könnten.

Während des Einlesens der Titeldatei werden die Daten des Grafikobjekts direkt gelesen, sobald der Dateiname bekannt ist. OBJECT.SHAPE weist die Definition dem Objekt Nummer 1 zu.

Stehen in der Titeldatei mehr Bitebenen als vom Programm erlaubt, verwendet das Videotitel-Programm nur die erlaubte Anzahl. Die Variable 'Farben1' liest den Wert aus der Datei. Ist 'Farben1' kleiner oder gleich dem Wert von 'Farben', übernimmt die Leseroutine die neue Anzahl in die Variable 'Farben'.

Soweit alles klar? Wie bitte? Ach so, stimmt ja: Den REM-Befehl kennen Sie ja auch noch nicht. Ihn haben wir in 'Titel-Speichern:' eingesetzt, um Ihnen eine weitere Verständnishilfe zu bieten. REM hat in AmigaBASIC ausnahmsweise nichts mit Strahlenbelastung und Tschernobyl zu tun. Es kommt vielmehr vom englischen Wort "Remark" und bedeutet "Anmerkung". Sie

können an jede Programmzeile eine Anmerkung anhängen. Wie wir es zum Beispiel in der ersten Zeile nach OPEN DatName$... gemacht haben. Dort steht:

```
REM Anzahl Textzeilen
```

Wir machen die Anmerkung, daß in der Variablen 'Zeilen' die Anzahl der Textzeilen steht. Wann immer AmigaBASIC auf den Befehl REM trifft, kümmert es sich nicht mehr um den Text, der dahinter steht. Sie können dort hinschreiben, was Sie wollen. Aber Vorsicht: Auch Befehle, die hinter REM folgen, werden nicht mehr erkannt. Probieren Sie im BASIC-Window:

```
FOR x=1 TO 10 : REM Anmerkung : NEXT x
```

Diese Schleife wird nicht korrekt ausgeführt, weil AmigaBASIC das NEXT x wegen dem vorangestellten REM-Befehl schlicht übersieht. Ein "FOR without NEXT"-Error ist die Folge. Um den gleichen Effekt wie durch REM zu erreichen, können Sie auch ein Hochkomma verwenden. Verwechseln Sie dieses Zeichen aber bitte nicht mit dem Apostroph. Sie erreichen das Hochkomma auf den Tastaturen der Amigas 500 und 2000 durch die Tastenkombination <ALT> <ä>. Wenn Sie neben ein Hochkomma mal versuchsweise einen Apostroph schreiben (Taste <'> und danach die Leertaste), erkennen Sie sicher den kleinen aber feinen Unterschied.

Vor dem ' muß nicht einmal ein Doppelpunkt stehen. Sehen wir uns eine Zeile aus der 'TitelSpeichern:'-Routine an:

```
PRINT #1,Eingel        ' Objekt eingelesen?
```

Hinter ' steht eine Anmerkung. Sie erklärt, daß die Variable 'Eingel' darüber Auskunft gibt, ob ein Objekt eingelesen wurde oder nicht. Um bestimmte Programmroutinen zu erklären oder auf Besonderheiten hinzuweisen, sind REM und ' hervorragend geeignet. Wir haben diese Möglichkeit in 'TitelSchreiben:' benutzt, um Ihnen zu erklären, welche Bedeutung die einzelnen Variablen haben.

Die Anmerkungstexte werden von AmigaBASIC nicht ausgeführt, müssen aber trotzdem gelesen und bearbeitet werden. Dafür wird Zeit benötigt. In zeitkritischen Programmteilen haben die Anmerkungen deshalb nichts verloren. Zwar dauert die Bearbeitung einer REM-Anweisung nur Sekundenbruchteile, jedoch in einer FOR...NEXT-Schleife, die vielleicht 8000mal durchlaufen wird, läppert sich diese Zeit schnell zu einer halben Minute und länger zusammen.

Die Unterprogramme sind eingegeben. Was jetzt noch fehlt, wissen Sie sicher schon selbst: Die neuen Programmpunkte müssen in die 'Auswahl:' aufgenommen werden. Fügen Sie die nächsten beiden Zeilen unterhalb von Punkt 6 ein:

```
PRINT "7 Titelsequenz laden"
PRINT "8 Titelsequenz speichern"
```

Das 'Abfrage:'-Programm muß an die beiden neuen Optionen angepaßt werden. Nach allen erforderlichen Arbeiten sieht dieser Programmteil so aus:

```
Abfrage:
  LOCATE 13,1
  PRINT "Ihre Wahl":;
  INPUT a$
  a$=LEFT$(a$,1)
  IF a$<"1" OR a$>"8" THEN BEEP : GOTO Abfrage
  IF a$="1" THEN Texteingabe
  IF a$="2" THEN ObjEinlesen
  IF a$="3" THEN ObjBewegung
  IF a$="4" THEN Farbdefinition
  IF a$="5" THEN Wiedergabe
  IF a$="6" THEN BildVorbereiten
  IF a$="7" THEN TitelLaden
  IF a$="8" THEN TitelSpeichern
  PRINT "Punkt "a$" nicht vorhanden"
  GOTO Abfrage
```

Die Bedienung ist auch ganz einfach: Wenn Sie einen Titel zusammengestellt haben und ihn nach der Wiedergabe (Punkt 5) für gut befinden, können Sie ihn mit Punkt 8 abspeichern. Das Unterprogramm fragt Sie nach dem gewünschten Dateinamen. In die angegebene Datei werden alle wichtigen Daten des Videotitels geschrieben. Wenn Sie später den Titel wieder einlesen wollen, wählen Sie Option 7 und geben den Dateinamen ein. Nach dem Lesen der Datei sind dem Programm alle Daten genauso bekannt, als ob Sie sie vorher von Hand festgelegt hätten, und Sie können "Wiedergabe" wählen.

Zu den Titelsequenzen noch eine Anregung: Wenn Sie die Programmteile 'Vorbereitungen:', 'Wiedergabe:' bis einschließlich 'GeschwBerechnen:' und 'TitelLaden:' außerdem die SUB-Routine 'Laden' zu einem kleinen Programm zusammensetzen, indem Sie den Rest des Videotitel-Programms rauswerfen, kommt ein Wiedergabe-Programm heraus, das Sie zum Beispiel für Titel vor eigenen Programmen einsetzen können. Speichern Sie dieses Programm aber bitte unbedingt unter neuem Namen ab, damit Sie das Videotitel-Programm nicht löschen. Wie Sie erreichen, daß ein Programm (wie das Wiedergabe-Programm) ein anderes Programm (z.B. Ihr Hauptprogramm) nachlädt und ausführt, erfahren Sie beim CHAIN-Befehl im Anhang B. Sie können dann einen Titel mit dem Videotitel-Programm erzeugen und ihn vielleicht vor einem selbstgeschriebenen Spiel laufen lassen. Natürlich auch vor Ihrer Finanz-Daten-Verwaltung, ganz wie Sie wollen.

4.7 BASICs kleine Bastelstunde - wir bauen unsere Befehle selbst

Wir haben jetzt einige Möglichkeiten kennengelernt, die IFF-Lade-Routine in vorhandene Programme einzubauen. Trotzdem wäre natürlich für Sie und auch für uns alles viel einfacher gewesen, wenn Microsoft von vornherein Befehle zum Laden und Speichern von Bildern in AmigaBASIC eingebaut hätte. Obwohl AmigaBASIC eine sehr leistungsfähige Programmiersprache ist, werden Sie wahrscheinlich beim Entwickeln eigener Programme öfter an den Punkt kommen, wo Sie eine bestimmte Funktion oder einen bestimmten Befehl vermissen, der zwar unheimlich nützlich wäre, aber eben leider nicht existiert. Man muß allerdings fairerweise zugeben, daß die Wünsche von Programmierern sehr breit gestreut sein können. Während wir als Grafik-Fans gern Befehle zum Laden und Speichern von IFF-Dateien hätten, sagt sich ein begeisterter Finanzmathematiker vielleicht: "Auf diese Spielereien kann ich gut verzichten, aber ein Befehl, der den Korrelationskoeffizienten einer Reihe statistischer Daten ermittelt, der wäre wirklich wichtig..."

Über Geschmack läßt sich ja bekanntlich streiten. Das wußten auch die Entwickler von AmigaBASIC und haben einige Möglichkeiten vorgesehen, wie man beim Programmieren eigene Routinen nachrüsten kann, die sich fast wie BASIC-Befehle verhalten.

Eine Methode, um das zu erreichen, haben Sie vor kurzem kennengelernt: Die SUB-Routinen. Vielleicht können Sie sich noch nicht so ganz vorstellen, wie wir mit dem SUB-Befehl neue Befehle programmieren wollen. Am besten, wir zeigen Ihnen mal ein Beispiel: Erinnern Sie sich noch an die Funktion DATE$? Wir haben ihre Bekanntschaft im Kapitel 1.17 bei der Entwicklung unseres Videotitel-Programms gemacht.

```
? date$
```

liefert das Datum, das AmigaDOS für das Tagesdatum hält. Wenn dieses Datum nicht stimmt, geben Sie bitte nicht Ihrem Amiga die Schuld. Schließlich sind Sie dafür verantwortlich, daß

in Preferences bzw. in die batteriegepufferte Echtzeituhr das richtige Datum angegeben wird. Auf jeden Fall bleibt bei DATE$ ein kleines Problem: Die amerikanische Schreibweise. AmigaBASIC zeigt nämlich zuerst den Monat an, dann den Tag und dann das Jahr. Vielleicht brauchen Sie aber in einem Programm häufig das Datum in deutscher Schreibweise (Tag, Monat, Jahr). Dann empfehlen wir Ihnen, folgende SUB-Routine hinter Ihr Programm zu hängen:

```
SUB Datum STATIC
  PRINT MID$(DATE$,4,2)"."LEFT$(DATE$,2)"."RIGHT$(DATE$,4)
END SUB
```

Dank diesem SUB-Programm können Sie ab jetzt in Ihrem Programm anstelle von PRINT DATE$ einfach

```
Datum
```

schreiben. Oder CALL Datum, wenn's Mißverständnisse gibt. Und schon wird das Datum an die aktuelle Cursorposition gedruckt. Während DATE$ als Ergebnis zum Beispiel "02-28-1988" lieferte, erhalten Sie durch unsere SUB-Routine "28.02.1988". Gut, nicht?
Auch im Direktmodus können Sie die SUB-Programme verwenden. Geben Sie doch einfach mal

```
Datum
```

im BASIC-Window ein. Wenn irgendwo im aktuellen Programm eine SUB-Routine steht, kann sie auch aus dem Direktmodus aufgerufen werden.

Jetzt verstehen Sie sicher auch, warum AmigaBASIC so oft die Fehlermeldung "Undefined subprogram" bringt (sie bedeutet: "Das SUB-Programm wurde nicht definiert"), wenn es eine Eingabe nicht versteht. Meist ist ein Tippfehler die Ursache dafür. Haben Sie zum Beispiel

```
pint a$
```

anstelle von

```
print a$
```

getippt, sucht AmigaBASIC verzweifelt nach einem SUB-Programm namens 'pint'. Und weil es keines findet, kommt die genannte Fehlermeldung. Probieren Sie das ruhig mal aus.

Jetzt nochmal zurück zu unserer 'Datum'-Routine: Kennen Sie überhaupt schon den MID$-Befehl? Im Gegensatz zu LEFT$ und RIGHT$ ermöglicht er, Zeichen aus der Mitte eines Strings zu isolieren. Sie geben dazu den Stringnamen an, die Position innerhalb des Strings und die Länge des Teils, den Sie isolieren wollen:

PRINT MID$(Stringname,Position,Länge)

Wenn Sie aus dem String 'a$' ab dem 4. Zeichen einen 2 Zeichen langen String isolieren wollen, erreichen Sie das mit MID$(a$,4,2). In unserer SUB-Routine brauchen wir MID$, um an die beiden Zeichen zu kommen, in denen der Tag steht (z.B. 28). Der zwei Zeichen lange LEFT$ liefert dann den Monat (10) und der vier Zeichen lange RIGHT$ das Jahr (1986).

Wer bisher gut aufgepaßt hat, wird jetzt fragen: "Bei vielen Befehlen gibt es doch noch eine ganze Latte Parameter hinten dran. Kann ich denn sowas auch in einer SUB-Routine erreichen?" Sie können. AmigaBASIC hat nämlich auch dafür vorgesorgt.

Zuerst wieder ein einfaches Beispiel: Sie stehen kurz vor Ihrem Amerika-Urlaub und brauchen deshalb in einem Programm eine Routine, die Dollar-Beträge in DM umrechnet. Oder Sie kommen gerade aus Amerika und fragen sich, wo nur Ihr ganzes Geld geblieben ist. Zunächst zum SUB-Programm. Wir gehen von einem Dollarkurs von 1.70 DM aus. Zum Zeitpunkt, als dieses Buch geschrieben wurde, war dieser Wert ziemlich optimistisch. Aber so etwas kann sich ja schnell ändern.

```
SUB Dollar (Wert) STATIC
  PRINT Wert "$ sind" Wert*1.70 "DM"
END SUB
```

Die Parameter, die einer SUB-Routine übergeben werden sollen, stehen in der SUB...STATIC-Zeile in Klammern hinter dem Namen der Routine. In unserem Fall hat der Paramter den Namen 'Wert'. Beim Aufruf können Sie die Klammern um den Wert weglassen. Probieren Sie also jetzt im BASIC-Window:

```
Dollar 34.00
```

Geben Sie die Zeile bitte genauso ein, wie wir sie abgedruckt haben. Ihr Amiga wird Ihnen antworten:

```
34 $ sind 57.8 DM
```

Hinter 'Dollar' haben Sie in unserem Beispiel den Dollar-Betrag 34.00 angegeben. Das SUB-Programm weist den übergebenen Wert der Variablen 'Wert' zu und errechnet daraus den DM-Wert. Die Variable 'Wert' ist eine lokale Variable des SUB-Programms. Im Hauptprogramm und im Direktmodus können Sie ihren Inhalt nicht feststellen. Das war in unserem Fall auch nicht nötig, denn das SUB-Programm druckt den umgerechneten Wert ja gleich aus.

Was aber tun, wenn Sie mit dem SUB-Programm einen Wert nur umrechnen und nicht sofort ausdrucken wollen? Auch das ist nicht allzu schwer: Diesmal rechnen wir zur Abwechslung von DM in Dollar um. Das Unterprogamm sieht dann so aus:

```
SUB Dollar (Wert) STATIC
  Wert = Wert * 1/1.70
END SUB
```

Und der Aufruf im BASIC-Window so:

```
Wert=10.20 : Dollar Wert : ? Wert
```

Als Ergebnis erhalten Sie 6. Denn 10.20 DM sind 6.00 $. Wie hat das nun genau funktioniert? Der Variablen 'Wert' haben Sie im Direktmodus den Wert 10.20 zugewiesen. Das ist der DM-Betrag. Dann haben Sie das SUB-Programm 'Dollar' mit dem Parameter 'Wert' aufgerufen. Bitte auch diesmal 'Wert' unbedingt ohne Klammern schreiben! Als Sie schließlich die Variable 'Wert' mit PRINT drucken ließen, hatte sie den neuen Inhalt 6. Wichtig ist, daß die Variable 'Wert' im Direktmodus und die Variable 'Wert' im SUB-Programm nichts miteinander zu tun haben. Sie hätten die Umrechnung auch mit jeder anderen Variablen aufrufen können. Wir treten sofort den Beweis an:

```
DM=10.20 : Dollar DM : ? DM
```

Das hat nichts mit Zauberei zu tun, sondern hängt mit der Arbeitsweise von SUB-Programmen zusammen: Die lokale Variable 'Wert' in der SUB-Routine 'Dollar' hat nur die Funktion, sich den Wert zu merken, der als Parameter mitgeschickt wurde, und ihn innerhalb der Routine für Berechnungen bereitzustellen. Wenn Sie hinter 'Dollar' direkt eine Zahl (wie 34.00 in unserem Beispiel von vorhin) angeben, war das auch schon alles. Geben Sie aber eine Variable (wie 'DM' in unserem Beispiel von gerade eben) an, die umgerechnet werden soll, wird dieser Variablen nach der Rückkehr aus der SUB-Routine der neue, errechnete Wert zugewiesen. Und das ist ja die Zahl, die in der lokalen Variablen 'Wert' gespeichert war.

Für den Fall, daß Sie nicht ganz sicher sind, ob Sie das eben wirklich verstanden haben, noch ein Beispiel. Wir schreiben ein ganz einfaches SUB-Programm, das die übergebene Zahl verdoppelt:

```
SUB Verdopple (Zahl) STATIC
  Zahl=Zahl*2
END SUB
```

Rufen Sie diese Routine bitte im BASIC-Window folgendermaßen auf:

```
a=2 : Verdopple a : print a
```

Toll, nicht wahr? Jetzt sind wir schon soweit, daß wir mit unserem Amiga Deutsch reden können. Naja - nicht ganz, dahinter steckt ein kleiner Trick: Unser SUB-Programm hat den Namen 'Verdopple'. Der Befehl

```
Verdopple a
```

ruft das SUB-Programm 'Verdopple' auf und übergibt als Parameter den Wert der Variablen 'a'. Diesen Wert übergibt die 'Verdopple'-Routine ihrer Variablen 'Zahl'. Mit 'Zahl' wird nun innerhalb der Routine gerechnet. Bei der Rückkehr übergibt das SUB-Programm den Inhalt seiner Variablen 'Zahl' dann wieder an die Parameter-Variable 'a'. So kann das Hauptprogramm das Ergebnis der Berechnung erfahren. Von der lokalen SUB-Variablen 'Zahl' hat es ja nie etwas gewußt.

Natürlich gibt es auch Situationen, wo Sie das alles gar nicht wollen: "Da verleiht man nun seine schöne runde Variable an die SUB-Routine, und was bekommt man zurück? Einen völlig anderen Wert." Das ist ja immerhin fast so, als ob Sie einem Bekannten Ihre Ausgabe unseres AmigaBASIC-Buchs leihen und dann nach einiger Zeit Grünbert Gräsleins "Ökologie im Vorgarten" zurückbekommen. "Aber was hast Du denn? Du hast mir ein Buch geliehen, und Du bekommst ein Buch zurück..."

Was also tun, wenn Sie dem SUB-Programm zwar den Wert einer Variablen mitteilen, ihn aber nicht verändern lassen wollen? Ganz einfach: Nun kommt die große Stunde der Klammern. Die Klammern um einen Parameter wirken wie ein Schutz, sie verhindern Veränderungen des Inhalts.

Wenn Sie die 'Verdopple'-Routine folgendermaßen aufrufen, verändert sich der Wert von 'a' nicht:

```
a=2 : Verdopple (a) : print a
```

Die Variable 'a' hat vor und nach dem SUB-Aufruf den Wert 2. Zwar läuft das SUB-Programm 'Verdopple' ganz normal durch, aber es kann sein Ergebnis nicht an 'a' weitergeben, weil 'a'

durch die Klammern geschützt ist. Das Verdoppeln wird dadurch natürlich ziemlich sinnlos. Aber wenn wir zurück zu unseren Währungsumrechnungen gehen, sehen Sie den Vorteil:

```
SUB Dollar (Wert) STATIC
  PRINT Wert "DM sind";
  Wert=Wert * 1/1.70
  PRINT Wert "$"
END SUB
```

Wenn Sie dieses SUB-Programm wie gewohnt aufrufen,

```
DM=21.60 : Dollar DM
```

erfahren Sie, daß 21.60 DM umgerechnet run 12.70 $ sind. Aber schauen Sie sich jetzt mal den Wert von DM an:

```
? DM
```

Hat doch tatsächlich der Dollar die Deutsche Mark verdrängt. Der Inhalt von 'DM' wurde 'Wert' übergeben, 'Wert' änderte sich und wurde zurück an 'DM' übergeben. Resultat: 'DM' hat sich auch geändert. Innerhalb eines Programms stünde der alte Wert von 'DM' jetzt nicht mehr zur Verfügung.

So hätten Sie das vermeiden können:

```
DM=21.60 : Dollar (DM)
```

Das Ergebnis ist dasselbe, aber die 'DM' behält ihren Wert. Möglicherweise beginnen Sie, langsam Ihr Interesse am internationalen Kursgefüge zu verlieren. Das ist auch nicht schlimm, selbst Finanzgenies soll es manchmal so ergehen.

Damit Sie gleich wieder mehr Spaß haben, verraten wir Ihnen jetzt, warum wir uns in diesem Kapitel so lang und breit über SUB-Aufrufe ausgelassen haben: Wir wollen nämlich eine SUB-Routine entwickeln, die IFF-Bilder speichern kann. Wenn Sie

die hinter Ihre Programme hängen, dann können Sie mit einem einfachen Befehl jedes beliebige Bild auf Diskette speichern. Das ist doch was, oder?

Ein kleines Stolpersteinchen auf dem Weg zum Erfolg müssen wir aber noch aus dem Weg räumen, und zwar im nächsten Zwischenspiel.

Zwischenspiel 7: AmigaBASICs Gruselkabinett - Zahlensysteme

Vielleicht vorab ein paar Worte zum Team Rügheimer/Spanik. Wir erlauben Ihnen hiermit aus besonderem Anlaß einen Blick hinter die Kulissen. Unsere Arbeitsteilung beim Schreiben eines Buches sieht ganz einfach so aus: Hannes Rügheimer legt sein gesammeltes Programmiererwissen in die Waage, Christian Spanik seine Erfahrung von mehreren Jahren Dasein als Werbetexter. Die Ideen entwicklen wir meistens zusammen, in einer Art geistigem Ping Pong. So einfach ist das. Nachdem Hannes aber, just als wir an diesem BASIC-Buch gearbeitet haben, seinen Pflichten gegenüber Vater Staat nachzukommen hatte (Sie kennen das ja: "Geeeefreiter Rüüügheimerr...") und er außerdem in Würzburg wohnt, war es zum Teil nötig, daß Hannes gewisse Teile des Buches im stillen Kämmerlein schrieb und Christian sie erst später überarbeiten konnte. So fanden sich innerhalb des Textes immer wieder "Kommunikationsteile", die demjenigen, der gerade an dem Text arbeitete, bestimmte Gedanken, Ideen oder auch Probleme bei einzelnen Textabschnitten schilderten.

"Warum um alles in der Welt erzählen die mir das jetzt bloß?", werden Sie sich wahrscheinlich fragen. Der Grund ist dieses Zwischenspiel: Es gehört sicherlich zu den schwierigsten im Buch. Und es entstand in Würzburg. Als es dann schließlich in Düsseldorf ankam, fand Christian just an der Stelle, an der Sie, lieber Leser, jetzt sind, folgende Anmerkung:

"((!!! Lieber Chris! Dieses Kapitel ist wirklich ein bißchen gruselig. Zahlensysteme und viel Mathe! Das Zeug ist aber leider wichtig für die SUB-Programme. Versuch's bitte so nett wie möglich zu machen...))"

Um es gleich zu sagen: Wir versuchten beide, es so nett wie möglich zu machen. Ob uns das gelungen ist, müssen wir Ihrem Urteil überlassen. Auf jeden Fall haben wir versucht, Ihnen auch hier ein komplexes Thema möglichst einfach zu erklären. Wenn Sie also nicht bis in die letzten Tiefen dieses Kapitels vordringen, ist es nicht so schlimm. Der Rest kommt noch im Lauf der Erfahrung. Wirklich wichtig sind solche Zahlenspielereien für die Leute, die später SUB-Routinen programmieren müssen. Soweit alles klar? Dann lassen Sie also den Mut nicht sinken, und los geht's:

Manchmal kann AmigaBASIC ganz schön pingelig sein. Und zwar besonders dann, wenn es um so ernste Dinge geht wie die Programmierung eigener Befehle. Immerhin gehen wir damit ja schon ziemlich nah an seine innersten Funktionen... Wovon wir sprechen? Nun, bei den verschiedenen Beispielen mit Dollars, Daten, Deutschen Mark und doppelten Zahlen haben wir Sie immer gebeten, auch ganze Beträge (wie 34 $) als Kommazahl anzugeben. Also etwa so:

```
Dollar 34.00
```

Warum eigentlich? Schließlich müßte doch die Zahl 34 genau dasselbe bewirken wie 34.00. Oder etwa nicht?

Schreiben Sie bitte noch einmal die Ur-Version des SUB-Programms 'Dollar' ins LIST-Window:

```
SUB Dollar (Wert) STATIC
  PRINT Wert "$ sind" Wert*2.25 "DM"
END SUB
```

Rufen Sie es dann im BASIC-Window auf:

```
Dollar 34
```

Was Ihr Amiga dazu meint, haben Sie wahrscheinlich gerade selbst erlebt: "Type mismatch"-Error meint er. "Typ stimmt nicht überein." Wie bitte? Wer oder was stimmt da nicht überein? Der Typ? Na, und er soll die 34 Dollar ja nicht heiraten, sondern umrechnen - um ehrlich zu sein, wir mußten auch ganz schön lange suchen, bis wir herausfanden, was Amiga jetzt schon wieder hatte. Wie gesagt: Er kann ganz schön pingelig sein, wenn's um seine innersten Funktionen geht.

Das Problem: Die Zahl 34 ist für den Amiga eine ganze Zahl. "Logisch, für uns doch auch?!" - Stimmt schon, aber die Variable 'Wert' ist nicht für ganze Zahlen gedacht. 'Wert' ist vielmehr eine Fließkommavariable. Vom "springenden Punkt" hat ja jeder schon mal gehört. Vom "hüpfenden Komma" wahrscheinlich auch. Aber "fließendes Komma"?

Nur die Ruhe! Der Bildschirm Ihres Amiga wird nicht langsam dahinschmelzen. Das ist alles ganz, ganz anders gemeint. Das, womit wir uns jetzt beschäftigen müssen, ist die interne Verwaltung von Zahlen bei AmigaBASIC. Vereinzelt haben wir ja schon verschiedene Zahlentypen kennengelernt, wie zum Beispiel die Integer-Zahlen, die 16-Bit-Zahlen oder die 32-Bit-Zahlen. Jetzt wird es Zeit, in das Ganze ein wenig System zu bringen. Kümmern wir uns am besten erst mal um's rutschende Komma oder wie das Ding heißt.

Fließkommavariablen sind die Variablen, mit denen Sie wahrscheinlich am häufigsten zu tun haben. Alle Variablennamen, die keine besondere Kennzeichnung am Ende haben, sind Fließkommavariablen. Also zum Beispiel 'Hallo', 'a', 'Farben' usw. Der Ausdruck "Fließkomma" bedeutet, daß das Komma nicht an einem festen Platz steht. Ein paar Beispiele für Fließkommawerte machen das deutlich:

```
100.23
3.141593
1.4
.143
4.1165
```

Das Komma steht an den verschiedensten Stellen innerhalb der Zahl, es ist nicht an seinen Platz gebunden. Alle gezeigten Zahlen können Sie in Fließkommavariablen ('Hallo', 'a', 'Farben' usw.) speichern. Diese Werte haben bis zu sieben Stellen. Mehr ist nicht drin. Probieren Sie's aus:

```
a=0.2435475776443 : ? a
```

Als Ergebnis erhalten Sie die Zahl .2435476

Nach sieben Stellen wurde die Kommazahl abgeschnitten und gerundet. Das Komma zählt bei den sieben Stellen übrigens nicht mit.

Das war eine ziemlich kleine Zahl. Und wie verhält sich Amiga-BASIC bei sehr großen Zahlen?

```
a=3426478236487367489 : ?a
```

Sie müssen übrigens nicht jedesmal genau dieselben Zahlen eintippen wie wir. Das wäre doch ein bißchen viel verlangt. Geben Sie einfach eine Zahl ein, die ungefähr gleich lang ist. Als Ergebnis unserer letzten Eingabe erhielten wir:

```
3.426478E+18
```

Diese Schreibweise für Zahlen kennen Sie vielleicht von Ihrem Taschenrechner. Man nennt sie "wissenschaftliche Darstellung" oder "Exponentialdarstellung". Der erste Name kommt nicht daher, daß sich um ihr Verständnis eine eigene Wissenschaft rankt. So schlimm wird's nun auch wieder nicht. Nein, Wissenschaftler (besonders Physiker und Mathematiker) haben berufsbedingt mit

unvorstellbar großen oder unvorstellbar kleinen Zahlen zu tun. (Unvorstellbar, was?) Deshalb haben sie sich eine besondere Schreibweise dafür einfallen lassen:

3.426478E+18 bedeutet ausführlich $3.426478 * 10^{18}$.

So erkennt man besser, in welcher Größenordnung sich die Zahl bewegt.

Im Zwischenspiel 4 haben Sie gelesen, daß das Binärsystem auf der Zahl 2 basiert. Bei jeder neuen Zweier-Potenz wird eine neue Stelle eröffnet: $2^0=1$ (bin. 1), $2^1=2$ (bin. 10), $2^2=4$ (bin. 100) usw. Das Dezimalsystem, also unser gewohntes Zahlensystem, funktioniert genauso. Nur daß es diesmal Zehnerpotenzen sind, die eine neue Stelle hervorbringen: $10^0=1$, $10^1=10$, $10^2=100$, $10^3=1000$. Und so geht's immer weiter. Mit jeder Zehnerpotenz werden die Zahlen größer. Nur wenn Sie's interessiert: Die Zahl 10^{18} aus dem Beispiel oben ist eine Trillion, das ist eine 1 mit 18 Nullen hinten dran. Also eine ziemlich große Zahl. Der Exponent (die Hochzahl) gibt bei den Zehner-Potenzen immer die Anzahl der Nullen an.

1.0E+3 ist $1.0 * 10^3$, also 1000. Drei Nullen.
3.4E+2 ist $3.4 * 10^2$ ist $3.4 * 100$ (zwei Nullen), also 340.

Auch für sehr kleine Zahlen benutzt AmigaBASIC die Exponentialdarstellung:

```
a= 1/202 : ? a
```

Das Ergebnis 4.950495E-03 bedeutet: $4.950495 * 10^{-3}$. 10^{-3} ist $1/10^3$. In normaler Schreibweise heißt die Zahl: 0.004950495

Bitte machen Sie sich keine Sorgen: Wenn Sie normalerweise mit solchen Zahlen nichts zu tun haben, werden Sie wahrscheinlich auch keine Programme schreiben, wo sie vorkommen. Und dann macht's nichts, wenn Sie das gerade nicht so hundertprozentig verstanden haben.

Das Ergebnis vom letzten Beispiel stimmt sowieso nicht genau, AmigaBASIC hat sich ein wenig verrechnet. "Jaja, der Amiga hat wirklich menschliche Züge - ein bißchen unordentlich, ein bißchen pingelig, und rechnen kann er auch nicht."

Aber lassen Sie sich erklären, woran's liegt. Im allgemeinen erwartet man von einem Computer ja nicht gerade, daß er Rechenfehler macht. Unser Amiga ist aber mit seinem Problem nicht allein. Alle Computer haben Schwierigkeiten mit Kommazahlen. Nach wie vor muß ja jede Zahl in Bits umgewandelt werden, bevor der Computer damit auch nur das Geringste anfangen kann. Wie das bei ganzen Zahlen geht, haben Sie im Zwischenspiel 4 gesehen. Bei Kommazahlen ist es wesentlich schwieriger, wir wollen hier nicht darauf eingehen. (Wer's unbedingt wissen will, kann sich ja mal mit dem Anhang D des AmigaBASIC-Handbuchs von Commodore beschäftigen. Aber sagen Sie nicht, wir hätten Sie nicht gewarnt.) Bei dieser Umwandlerei von Zahlen in Bits und zurück schleichen sich kleinere Rechenfehler ein. Meistens handelt es sich um Rundungsfehler. Versuchen Sie zum Beispiel mal

```
? 100.1 - 100
```

Statt dem richtigen Ergebnis 0.1 kommt AmigaBASIC auf den Wert 9.999847E-02. Das ist 0.099, also ziemlich nahe dran. Aber ganz genau stimmt's eben nicht.

Apropos "genau". Die Beispiele, die wir bisher für Fließkommazahlen kennengelernt haben, waren alles Zahlen "einfacher Genauigkeit". Wenn es Zahlen "einfacher Genauigkeit" gibt, muß es wohl noch genauere Zahlen geben. (Was auch immer das bedeuten mag.)

Die "Genauigkeiten" hängen mit der internen Zahlenverwaltung zusammen: Einfach genaue Zahlen können bis zu sieben Stellen haben. Die Werte liegen in der Größenordnung von 10^{-38} und 10^{+38}. Die kennen Sie also schon. Im Gegensatz dazu gibt es die doppelt genauen Zahlen. Sie haben bis zu 16 Stellen und können Wert zwischen 10^{-308} und 10^{+308} beinhalten. Normalerweise verwendet AmigaBASIC Fließkommavariablen einfacher Genauig-

keit. Das sind die Variablen vom Typ 'Hallo', 'a' und 'Farben', aber das kennen Sie ja schon... Wenn Sie mit Variablen doppelter Genauigkeit arbeiten wollen, müssen Sie die Variablen besonders kennzeichnen. Das Zeichen für doppelte Genauigkeit ist ein #-Zeichen hinter dem Variablennamen.

Vergleichen Sie:

```
a=1/202 : ? a
```

(Ergebnis: 4.950495E-03) und

```
a#=1/202 : ? a#
```

(Ergebnis: 4.950494971126318D-03). Die zweite Zahl hat viel mehr Stellen, also ist sie genauer. Das D, das dort auftaucht, wo wir ein E gewohnt sind, weist darauf hin, daß es sich um eine doppelt genaue Zahl handelt (engl.: "Double Precision Number"). Sonst bleibt aber alles beim alten. 1.0E+3 und 1.0D+3 bedeuten dasselbe, nämlich 10^3, also 1000.

Wahrscheinlich fragen Sie sich, ob es überhaupt jemanden gibt, der sich für soviele Stellen bei einem Rechenergebnis interessiert. Ob Sie die höhere Genauigkeit brauchen, hängt von Ihren Berechnungen ab. Wenn möglich, sollten Sie lieber auf doppelt genaue Zahlen verzichten. Sie haben nämlich auch Nachteile: Die Berechnungen dauern länger, und die Zahlen brauchen mehr Speicherplatz.

Jetzt wäre es wirklich an der Zeit, daß Sie eine kleine Pause einlegen. Die ganze Rechnerei war ja doch ziemlich anstrengend. Aber den schlimmsten Teil haben Sie sowieso überstanden. Komplizierter wird's nicht mehr. Es ließ sich leider nicht vermeiden, Ihnen das alles zu erzählen, denn in manchen Dingen nimmt's der Amiga sehr genau. (Eben doppelt genau.)

Haben Sie ein wenig verschnauft? Dann wollen wir jetzt ein bißchen weitermachen. Nochmal zur Wiederholung: Bisher haben Sie Fließkommazahlen einfacher und doppelter Genauigkeit kennengelernt. Die einfach genauen sind die normalen Variablen,

die von AmigaBASIC verwendet werden. Die doppelt genauen werden beim Variablennamen durch ein #-Zeichen gekennzeichnet.

Sie haben ja selbst gemerkt, daß Kommazahlen dem Amiga nicht unbedingt leichtfallen. Deshalb vermeidet er sie auch lieber, wenn's nur irgendwie geht. Da legt unser Computer die Mentalität jedes mittelmäßigen Mathe-Schülers an den Tag. Wenn also Zahlen benutzt werden, die mit Sicherheit keinen Nachkommateil haben, sind die Integer-Variablen dem Amiga viel lieber. Sie wissen ja noch: Variablen wie 'Hallo%', 'a%' und 'Farben%' können ganzzahlige Werte zwischen -32768 und +32767 speichern. Also 16-Bit-Zahlen mit einem Vorzeichen-Bit. Für viele Werte (wie Angaben über Farben, Koordinaten, Geschwindigkeiten...) ist dieser Zahlenbereich völlig ausreichend. Und wenn die Zahlen mal größer werden, aber immer noch keine Kommastellen haben, nimmt sich AmigaBASIC einfach noch ein paar Bits dazu: 32-Bit-Zahlen werden verwendet. Sie können Werte zwischen -2147483648 und +2147483647 beinhalten. So groß werden die Zahlen beim Programmieren aber selten.

Wenn eine Variable eine Integerzahl mit 32 Bit speichern soll, müssen Sie an den Variablennamen ein &-Zeichen anhängen. Schauen Sie sich mal den Unterschied an:

```
a%=100000
```

gibt einen Error: "Overflow" - "Die Zahl ist zu groß." Probieren Sie mal:

```
a&=100000
```

So gibt es damit keine Probleme mehr.

Sie haben beim Eingeben unserer Programme manchmal gesehen, daß AmigaBASIC 32-Bit-Zahlen im LIST-Window automatisch mit einem &-Zeichen kennzeichnet. Zum Beispiel hat der Amiga beim Malprogramm an die Zahl 65535 beharrlich sein &

gehängt. Kunststück, denn diese Zahl kann mit 15 Bit (eines geht ja immer fürs Vorzeichen drauf) nicht gespeichert werden. Also braucht der Amiga 32 Bit, und das zeigt er durch das &.

Jetzt haben Sie's schon fast geschafft. Die letzte Variablenart, die AmigaBASIC kennt, macht uns keine Schwierigkeiten mehr: Es sind die Stringvariablen. Sie beinhalten Zeichenketten (die übrigens bis zu 32767 Zeichen lang sein dürfen, 32767 ist eine von Amigas Lieblingszahlen) und werden durch ein $-Zeichen gekennzeichnet.

Noch mal alles in einer Übersicht:

Zahlentyp	Kennzeichnung der Variablen	Speicherbedarf	Beispiele für Variableninhalt
Fließkomma, einfach genau	keine oder !	4 Bytes	.3245643, 2.435E+09
Fließkomma, doppelt genau	#	8 Bytes	4.901960957795382D-03
kurze Integerzahl	%	2 Bytes=16 Bit	32767, -17
lange Integerzahl	&	4 Bytes=32 Bit	-2147483648, 65535
String	$	5 Bytes + Länge des Strings	"Christian", "Brigitte"

Tabelle 9: Die Zahlentypen in AmigaBASIC

Durch die Kennzeichnung am Variablennamen entstehen völlig unabhängige Variablen. Daß 'a' und 'a$' nichts miteinander zu tun haben, ist für uns mittlerweile selbstverständlich. Aber auch 'a#', 'a%' und 'a&' können völlig verschiedene Inhalte haben. Nur mit dem Ausrufezeichen müssen Sie aufpassen: Normalerweise werden Fließkommavariablen einfacher Genauigkeit über-

haupt nicht gekennzeichnet (das sind ja die Standardvariablen von AmigaBASIC). Wenn Sie doch ein "!" anhängen, ist das nur eine formale Angelegenheit und bewirkt keine Unterscheidung: 'Hallo' und 'Hallo!' haben immer denselben Inhalt.

Nach alledem haben wir aber immer noch nicht erklärt, warum der Amiga bei unseren SUB-Aufrufen eine Zahl wie 34 nicht so ohne weiteres akzeptiert. Schließlich gibt es ja auch Fließkommazahlen, die keinen Teil mehr hinter dem Komma haben: 'Wert'=34 hat doch z.B. noch nie Probleme gemacht.

AmigaBASIC geht aber bei der Speicherung von Zahlen seinen ganz besonderen Vorlieben nach: Wenn in einer Eingabe eine ganze Zahl auftaucht, speichert sie AmigaBASIC intern als Integervariable. 34 ist eine ganze Zahl. Diese Integerzahl soll im SUB-Programm einer Fließkommavariablen übergeben werden. Und das geht nicht: "Type mismatch". Die Variablentypen stimmen nicht überein. AmigaBASIC kommt mit der internen Zahlenverwaltung durcheinander. Was kann man dagegen tun? Eine Möglichkeit ist, Ihren Amiga auszutricksen. Wie im letzten Kapitel können Sie einfach Nullen hinter dem Komma angeben (34.00). So geht's aber auch:

```
Dollar 34.
```

Durch den Punkt hinter der Zahl 34 denkt AmigaBASIC, es handle sich um eine Fließkommazahl und speichert den Wert auch in dieser Form. Nun macht die Übergabe keine Probleme mehr. Sie können eine Zahl auch auf einen bestimmten Typ festlegen, indem Sie die jeweilige Kennzeichnung anhängen:

```
Dollar 34!
```

zwingt AmigaBASIC, den Wert 34 als Fließkommazahl einfacher Genauigkeit zu behandeln. Ein paar weitere Beispiele: Aus 3.4% wird 3 (hat nichts mit Prozent zu tun...). Die Zahl, die normalerweise als Fließkommazahl gespeichert würde, wird eine 16-Bit-Integerzahl. Der Nachkommateil geht dabei verloren. Aus 0.4& wird 0 (dasselbe, aber mit 32 Bit). Aus 3# wird 3. Sie se-

hen der Zahl zwar nichts an, aber sie wird intern als Fließkommazahl doppelter Genauigkeit behandelt. So merkt man den Unterschied:

```
? 1/3
```

und

```
? 1#/3
```

Was aber, wenn Sie von vornherein wissen, daß ein Parameter nur ganzzahlig sein kann, und Sie keine Lust haben, jedesmal ein Kennzeichen anzuhängen? Dann kennzeichnen Sie einfach die zugehörige Variable im SUB-Programm als Integervariable:

```
SUB Warte (Sek%) STATIC
Warten:
  TIM=INT(TIMER)
  WHILE INT(TIMER)=TIM : WEND
  Sek%=Sek%-1
  IF Sek%>0 THEN Warten
END SUB
```

Dieses SUB-Programm wartet eine angegebene Anzahl an Sekunden. Die Routine selbst kennen Sie aus dem Wiedergabeteil des Videotitel-Programms. Die Variable 'Sek%' erwartet bei der Paramterübergabe einen Integerwert. Durch den Aufruf

```
Warte 10
```

wird ihr genau so ein Wert übergeben. Das SUB-Programm wartet 10 Sekunden. Wenn Sie diese Routine aufrufen, dürfen Sie diesmal natürlich keine Kommazahlen angeben, sonst tritt das alte Problem wieder auf, nur mit anderen Variablentypen: Ein Fließkommawert kann keiner Integervariablen zugewiesen werden.

So. Jetzt haben Sie ein schönes Stück Arbeit hinter sich. Dafür kennen Sie nun auch die innersten Geheimnisse von Amiga-

BASIC. Mit diesem Wissen steht unserem nächsten Projekt nichts mehr im Weg: Wir bauen eine eigene SUB-Routine fürs Speichern von IFF-Bildern.

4.8 Rettet die Bilder! - die PICSAVE-Routine

Hier ist jetzt die Belohnung für Ihre Geduld. Das folgende SUB-Programm stellt einen Befehl zum Abspeichern von Grafikdaten zur Verfügung.

```
SUB PICSAVE (Nam$,WindowNr%,FeldJN%) STATIC
  IF FeldJN%=1 THEN SHARED Farben%()
  IF FeldJN%=0 THEN
    IF Farben%(0,0)<>2 THEN ERASE Farben% : DIM Farben%(31,2)
    RESTORE Farbtabelle
    FOR x=0 TO 31
      READ Farben%(x,0),Farben%(x,1),Farben%(x,2)
    NEXT x
Farbtabelle:
    DATA 2,3,10, 15,15,15, 0,0,0, 15,8,0
    DATA 0,0,15, 15,0,15, 0,15,15, 15,15,15
    DATA 6,1,1, 14,5,0, 8,15,0, 14,11,0
    DATA 5,5,15, 9,0,15, 0,15,9, 12,12,12
    DATA 0,0,0, 13,0,0, 0,0,0, 15,12,10
    DATA 4,4,4, 5,5,5, 6,6,6, 7,7,7
    DATA 8,8,8, 9,9,9, 10,10,10, 11,11,11
    DATA 12,12,12, 13,13,13, 14,14,14, 15,15,15
  END IF
  IF Nam$="" THEN EXIT SUB
  AltWindowNr=WINDOW(1)
  WINDOW WindowNr%
  Breite=WINDOW(2)
    IF Breite>320 THEN
      Breite=640
      Auflsg=2
      Ebene=16000
    ELSE
      Breite=320
      Auflsg=1
```

```
    Ebene=8000
  END IF
Hoehe=WINDOW(3)
  IF Hoehe>200 THEN
    Hoehe=400
    Ebene=Ebene*2
    Auflsg=Auflsg+2
  ELSE
    Hoehe=200
  END IF
Farben=LOG(WINDOW(6)+1)/LOG(2)

OPEN Nam$ FOR OUTPUT AS 1 LEN=FRE(0)-500
  PRINT #1,"FORM";
  PRINT #1,MKL$(156+Ebene*Farben);
  PRINT #1,"ILBM";
  PRINT #1,"BMHD";MKL$(20);
  PRINT #1,MKI$(Breite);MKI$(Hoehe);
  PRINT #1,MKL$(0);
  PRINT #1,CHR$(Farben);
  PRINT #1,CHR$(0);MKI$(0);MKI$(0);
  PRINT #1,CHR$(10);CHR$(11);
  PRINT #1,MKI$(Breite);MKI$(Hoehe);

  PRINT #1,"CMAP";MKL$(96);
  FOR x=0 TO 31
    PRINT #1,CHR$(Farben%(x,0)*16);
    PRINT #1,CHR$(Farben%(x,1)*16);
    PRINT #1,CHR$(Farben%(x,2)*16);
  NEXT x

  PRINT #1,"BODY";MKL$(Ebene*Farben);
  Adr=PEEKL(WINDOW(8)+4)+8
  FOR x=0 TO Farben-1
    Ebnadr(x)=PEEKL(Adr+4*x)
  NEXT x
  FOR y1=0 TO Hoehe-1
    FOR b=0 TO Farben-1
      FOR x1=0 TO (Breite/32)-1
        PRINT #1,MKL$(PEEKL(Ebnadr(b)+4*x1+(Breite/8)*y1));
```

```
      NEXT x1
     NEXT b
     PAdr=Ebnadr(0)+(Breite/8)*y1
     POKE PAdr,PEEK(PAdr) AND 63
     POKE PAdr+Breite/8-1,PEEK(PAdr+Breite/8-1) AND 252
    NEXT y1

    PRINT #1,"CAMG";MKL$(4);
    PRINT #1,MKL$(16384);
  CLOSE 1
  WINDOW AltWindowNr
END SUB
```

Wenn Sie dieses Programm selbst eingetippt haben und abspeichern (was Sie ja gerade tun wollten, nicht wahr...), speichern Sie es bitte im ASCII-Format ab. Das heißt also mit:

```
save "Picsave",a
```

In dieser Form finden Sie es übrigens auch auf der beiliegenden Diskette.

Um den Eigenbau-Befehl PICSAVE in anderen Programmen benutzen zu können, muß die Routine ja mit MERGE angehängt werden. Und Sie erinnern sich bestimmt noch daran, daß dabei das anzuhängende Programm als ASCII-Datei vorliegen muß.

Sicher gibt es im Programm noch ein paar Dinge, die Sie nicht kennen oder nicht verstehen. Deshalb besprechen wir jetzt die einzelnen Programmschritte.

In der SUB...STATIC-Zeile steht der Name unserer Routine. Wir nennen sie 'PICSAVE'. (Ein Kürzel für "Picture Save" - das wäre ein schöner Name für den Befehl gewesen, wenn Microsoft ihn doch eingebaut hätte. Schnüff.) Hinter dem Namen stehen in Klammern die Übergabeparameter. Wie Sie sehen, kann man

auch mehrere Werte gleichzeitig übergeben. Bloß der Variablentyp muß übereinstimmen. Die notwendigen Angaben bei Aufruf von 'PICSAVE' sehen so aus:

PICSAVE "(Dateiname)",(Windownr.),(Angabe über Farbfeld)

Zum Dateinamen müssen wir Ihnen nicht mehr viel sagen. Sie können einen beliebigen Namen eingeben, evtl. vorher noch das Laufwerk bzw. den Diskettennamen und den Weg über die Unter-Inhaltsverzeichnisse.

An zweiter Stelle geben Sie die Nummer des Windows an, dessen Inhalt abgespeichert werden soll.

Und am Schluß kann eine 1 oder eine 0 stehen. Der Wert 1 bedeutet, daß Sie dem SUB-Progamm selbst ein Datenfeld namens 'Farben%' bereitstellen, das die Dimensionen (31,2) hat und - als Werte zwischen 0 und 15 - die RGB-Zusammnsetzung der verwendeten Farben beinhaltet. Wie so ein Feld erzeugt und benutzt wird, sehen Sie zum Beispiel in unserem Malprogramm. Der Wert 0 bedeutet, daß die Standardfarben der Workbench in der Datei abgespeichert werden sollen. So können Sie ohne große Vorbereitungen Grafik abspeichern, die Sie mit BASIC-Befehlen erzeugt haben. (Wir denken da zum Beispiel an unsere Sinus-Grafiken...)

Der letzte der drei Werte ('FeldJN%' steht für "Feld Ja/Nein") wird auch gleich in den nächsten beiden Zeilen verarbeitet. Haben Sie 1 angegeben, macht sich das SUB-Programm mit dem SHARED-Befehl die Inhalte des Feldes 'Farben%' zugänglich. Gibt es in Ihrem Hauptprogramm kein Feld namens 'Farben%', meldet AmigaBASIC einen "Undefined array"-Error ("Datenfeld wurde nicht definiert").

Wenn 'FeldJN%' den Wert 0 hat, gibt's ein bißchen mehr zu tun: Zuerst schaut das SUB-Programm im Feldelement 'Farben%(0,0)' nach, welcher Wert dort steht. Ist der Inhalt eine 2, wurde das Feld schon einmal definiert. In diesem Fall darf kein zweiter

DIM-Befehl für dasselbe Feld ausgeführt werden, das würde ja einen "Duplicate Definition"-Error verursachen: Datenfelder dürfen nur einmal dimensioniert werden.

Was aber, wenn das Feld noch nicht definiert war? Wenn Sie ein bestimmtes Datenfeld benutzen, ohne vorher ein DIM für dieses Feld ausgeführt zu haben, wird es von AmigaBASIC automatisch dimensioniert. Und zwar auf die Standardgröße von 10 Elementen. Ein Beispiel:

Geben Sie im BASIC-Window

```
? Testfeld(1)
```

ein. Ein Feld namens 'Testfeld' gibt es bisher noch nicht. Das erkennt AmigaBASIC und gibt sich selbst den Befehl DIM Testfeld(10). Nun können Sie die Feldelemente 'Testfeld(0)' bis 'Testfeld(10)' benutzen. Wenn Sie aber z.B. versuchen, auf das Element 'Testfeld(11)' zuzugreifen,

```
? Testfeld(11)
```

erhalten Sie einen "Subscript out of Range"-Error (Element außerhalb des erlaubten Bereichs). Dieser Fehler tritt immer dann auf, wenn Sie Feldelemente benutzen wollen, die außerhalb der festgelegten Dimensionierung liegen. Und das 'Testfeld' wurde ja bloß auf 10 dimensioniert. Alles klar?

Dasselbe geschieht auch bei zwei und mehr Dimensionen:

```
? NocheinTestfeld(1,1)
```

erzeugt ein automatisches DIM NocheinTestfeld(10,10). Genauso verhält sich AmigaBASIC, wenn wir in unserem Programm das Element 'Farben%(0,0)' abfragen: Gibt es noch kein Feld 'Farben%', wird es automatisch auf 'Farben%(10,10)' dimensioniert. Wir brauchen aber für unsere Farbwerte die Dimensionen 'Far-

ben%(31,2)'. Also bleibt uns in diesem Fall nichts anderes übrig, als das Feld wieder zu löschen und mit den richtigen Werten neu zu dimensionieren.

Zum Löschen eines Datenfelds dient der Befehl ERASE. Dahinter geben Sie einfach den Feldnamen an. Das Feld mit allen Inhalten wird gelöscht, und es kann später mit DIM neu dimensioniert werden. Das tun wir auch gleich danach: DIM Farben%(31,2).

Wozu das alles? Die Notwendigkeit der ganzen Trickserei haben wir mal wieder Microsoft zu verdanken. Wie schon mehrmals erwähnt, gibt es keinen Befehl, um die derzeitigen Farbwerte auszulesen. Wir brauchen aber irgendwelche Farbwerte für den CMAP-Chunk. Um Ihnen dieses Problem abzunehmen, stellen wir im SUB-Programm 'PICSAVE' eine Reihe von DATA-Zeilen zur Verfügung, in denen die RGB-Werte der Workbench-Standardfarben abgelegt sind. Das sind die Farben, die von BASIC automatisch benutzt werden, solange Sie keinen PALETTE-Befehl benutzen. Wenn Sie Farben in Ihrem Programm ändern und das entstandene Bild abspeichern wollen, bleibt Ihnen leider nichts anderes übrig, als in einem Datenfeld 'Farben%' selbst für die richtigen Werte zu sorgen.

Vor dem Lesen des Datenfelds brauchen wir noch einen neuen Befehl: RESTORE. Mit RESTORE können Sie den Zeiger, der bei READ auf das nächste DATA-Element zeigt, auf ein bestimmtes Label setzen. In unserem Fall 'Farbtabelle:'. So ist sichergestellt, daß das SUB-Programm seine eigenen Daten liest und nicht etwa die Inhalte von DATA-Zeilen, die zu Ihrem Hauptprogramm gehören. Wir erreichen Datenschutz und Farbschutz in einem: Erstens haben Ihre Daten im CMAP-Chunk einer IFF-Datei nichts verloren, und zweitens könnte kein IFF-Leseprogramm aus Daten wie "4000 Düsseldorf" irgendeine vernünftige Farbe produzieren.

Die darauffolgende FOR...NEXT-Schleife liest die RGB-Werte ins Feld 'Farben%'. Übrigens: Wenn Sie sich fragen, wie wir zu den Daten der Workbench-Farben gekommen sind - nur durch

Ausprobieren. Wir haben jede Farbe solange mit dem Original verglichen, bis beide übereinstimmten.

Wurde als 'Nam$' ein leerer String angegeben, soll die Routine abgebrochen werden. EXIT SUB ermöglicht jederzeit den Rücksprung aus einem SUB-Programm. Geht's aber normal weiter, merkt sich 'AltWindowNr' die Nummer des aktuellen Ausgabewindows. Diese Nummer erfahren wir mit WINDOW(1). Nach der Rückkehr ins Hauptprogramm wollen wir schließlich alles so zurückgeben, wie wir es vorgefunden haben, auch und gerade die Zuteilung der Windows. Um an einige wichtige Daten zu kommen, müssen wir das Window in der Variablen 'WindowNr%' zum aktuellen Window machen. Nun kommen ein paar neue WINDOW-Funktionen: WINDOW(2) liefert zum Beispiel die Breite des aktuellen Ausgabewindows. Diesen Wert erhält die Variable 'Breite'. Wir wollen immer Windows abspeichern, die die Größe eines ganzen Screens einnehmen. Die IF...THEN/ELSE/END IF-Struktur sorgt dafür, daß die Breite entweder auf 320 Punkte oder 640 festgelegt wird. Abhängig davon legen wir auch gleich die Auflösungsstufe fest (1 für 320 Punkte, 2 für 640 Punkte) und den Speicherbedarf pro Bitebene (8000 Bytes für 320 Punkte, 16000 Bytes für 640 Punkte). Dann erfahren wir durch WINDOW(3) die Höhe des aktuellen Windows. Auch sie wird entweder auf 200 oder 400 Pixels festgesetzt. Bei 400 Pixels (Interlace-Modus) verdoppelt sich außerdem der Speicherbedarf pro Bit-Ebene, und der Wert für 'Auflsg' erhöht sich um 2 (und ergibt somit 3 für niedrigauflösende Interlace-Bilder und 4 für hochauflösende).

Zu guter Letzt brauchen wir noch die Anzahl der Bitebenen. Auch sie läßt sich berechnen, allerdings nur über Umwege: WINDOW(6) ergibt die Anzahl der erlaubten Farben im aktuellen Ausgabewindow. Bei 3 Bitebenen wäre das z.B. die Zahl 7. Wir haben oft die Formel 'Maxfarbe=(2^Bitebenen)-1' benutzt, damit errechnet man die Anzahl der erlaubten Farben. Diesmal kennen wir diese Zahl und suchen dafür die Bitebenen. Wir müssen die Formel also umkehren. Wenn man weiß, daß $2^x=8$ ist, erhält man x durch die Formel LOG(8)/LOG(2). LOG steht für Logarithmus. Wenn Sie den vorher nicht kannten, müssen Sie ihn sich jetzt auch nicht merken. (Und hier wieder unser Bon-

bon für den Mathematiklehrer und seinen besten Schüler: Die BASIC-Funktion LOG errechnet den natürlichen Logarithmus zu Basis e=2.718282.)

Damit wären dann auch alle Vorbereitungen abgeschlossen. Wir schreiben jetzt die IFF-Daten in die angegebene Datei. Ein paar Anmerkungen hierzu: Für den FORM-Chunk erfahren wir den Gesamtspeicherbedarf durch 156+'Ebene'*'Farben'. 156 ist die Dateilänge ohne BODY, und BODY hat die Länge (Bytes pro Ebene)*(Anzahl der Ebenen). Diese Längenangabe steht auch zu Beginn des BODY-Chunks: 'Ebene'*'Farben'.

Nach dem CLOSE machen wir das Window 'AltWindowNr' wieder zum aktuellen Window (wir hatten uns ja vorgenommen, vor dem Rücksprung aufzuräumen), dann wird die SUB-Routine beendet.

Jetzt haben Sie Ihren persönlichen IFF-Speicherbefehl. Er heißt PICSAVE und kann mit MERGE an jedes Programm angehängt werden. Die Auflösungsstufe, die Breite und Höhe sowie die Anzahl der Bitebenen stellt die Routine selbst fest. Wollen Sie also z.B. den Inhalt des BASIC-Windows abspeichern, rufen Sie die SUB-Routine so auf:

```
PICSAVE "BASICWindow",1,0
```

Die entstandene IFF-Datei können Sie mit dem IFF-Leseprogramm aus Kapitel 4.3 jederzeit wieder einlesen. Um außerhalb von AmigaBASIC mit der IFF-Datei zu arbeiten, brauchen Sie - da es sich um ein Bild im 640 * 200-Modus handelt - ein Programm wie "Deluxe Paint", das im hochauflösenden Modus arbeitet. Mit so einem Malprogramm können Sie dann auch Hardcopies ausdrucken. Wenn Sie BASIC-Grafiken wie unsere Sinus-Flächen in Graphicraft bearbeiten wollen, müssen Sie die Grafiken in ein Window auf einem 320 * 200-Pixels-Screen erstellen.

Auch im Balken-/Tortengrafik-Programm gibt es keine großen Probleme, wenn Sie das SUB-Programm 'PICSAVE' einsetzen wollen. Hier eine kurze Einbauanleitung:

- Laden Sie das Statistikdaten-Programm.
- MERGEn Sie daran die "PICSAVE"-Routine.
- Nehmen Sie im Programmteil 'Vorbereitungen:' folgende Zeile in die MENU-Definitionen mit auf:

```
MENU 2,3,1,"speichern"
```

- Fügen Sie im Programmteil 'Menukontrolle:' unter der Zeile

```
IF Men=2 THEN
```

  die folgenden Zeilen ein:

```
In Menpunkt=3 THEN
  MENU 1,0,0 : MENU 2,0,0
  MENU OFF
    GOSUB Eingabename
  WINDOW 99
    PICSAVE Nam$,99,0
  WINDOW 1
  MENU ON
  MENU 1,0,1 : MENU 2,0,1
END IF
```

- Speichern Sie das entstandene Programm ab.

Das Ganze findet sich übrigens auch unter dem Namen "BTDaten.IFF" auf unserer Diskette im Buch.

Im "Grafik"-Pulldown gibt es jetzt die neue Option "speichern". Wenn Sie sie anwählen, wird der aktuelle Inhalt des Grafikwindows auf Diskette abgespeichert. Bedenken Sie bitte, daß für die entstandene Datei dasselbe gilt wie für alle 640*200-Pixels-

Grafiken - sie können nicht mit Graphicraft bearbeitet werden. Und in Programmen wie Deluxe Paint oder Aegis Images müssen Sie die höhere Auflösungsstufe anwählen. Wenn Sie ein geeignetes Programm besitzen, können Sie die Grafiken bearbeiten und ausdrucken, wie wir Ihnen vor einiger Zeit versprochen hatten.

Mit dem Vorrat an IFF-Routinen, den Sie jetzt besitzen, müßten Sie in der Lage sein, sich jeden weiteren Wunsch zum Thema Laden und Abspeichern von Grafiken selbst zu erfüllen. Wir wollen dieses Thema daher jetzt verlassen, und uns mit anderen interessanten Aspekten von "Daten" beschäftigen.

5. Und dann kann man noch... - Was man mit Daten noch alles anfangen kann

Um es gleich vorweg zu sagen: Alles, was man mit Daten anfangen kann, können wir Ihnen in den nächsten Kapiteln auch nicht vorführen. Schließlich lebt eine ganze Branche davon, sich täglich Neues auf diesem Gebiet einfallen zu lassen. Und die Branche lebt nicht schlecht. Aber wir zeigen Ihnen ein Reihe neuer BASIC-Befehle zum Thema Datenverarbeitung. Da fehlt hauptsächlich noch das Thema "relative Dateien". Es gibt nämlich neben sequentiellen Dateien noch eine zweite Art, Dateien auf Diskette anzuordnen. Aber dazu kommen wir gleich. Wenn Sie den nun folgenden Teil unseres Buchs gelesen haben und unsere Beispiele immer schön mittippen, sind Sie danach übrigens auch stolzer Besitzer eines recht leistungsfähigen Datenbank-Programms.

5.1 Frei nach Albert Einstein - relative Dateiverwaltung

Bekanntlich ist alles relativ - warum also sollte das nicht auch für Daten gelten? Seit Kapitel 3.4 arbeiten wir ja nur mit sequentiellen Dateien. Wenn Sie's vergessen haben sollten: "Sequentiell" bedeutet "hintereinander". Sie haben es vielleicht nicht immer gemerkt, aber seit wir diesen Dateityp kennen, haben wir ihn ständig in der einen oder anderen Weise benutzt. Dateien, die zum Drucker geschickt werden, sind sequentiell. (Logisch, denn ein Drucker druckt immer ein Zeichen nach dem anderen.) Dateien, die von der Tastatur einlesen oder auf den Bildschirm schreiben, sind ebenfalls sequentiell. Und auch bei den IFF-Dateien haben wir immer schön einen Chunk nach dem anderen geschrieben - eben in sequentielle Dateien.

Für all die genannten Beispiele sind sequentielle Dateien auch sehr sinnvoll. Wenn es aber um das Lesen und Schreiben von Adreßlisten, Schallplattensammlungen oder ähnlichen Daten geht, haben die sequentiellen Dateien einen großen Nachteil. Sie erinnern sich: "Sequentielle Dateiverarbeitung" bedeutet, daß alle Daten nacheinander auf die Diskette geschrieben werden. Und auch nacheinander wieder gelesen werden müssen. Wenn Sie die

Adresse von Frau Burgschmidt suchen, bleibt Ihnen nichts übrig, als vorher die Adressen von Herrn Dr. Pleticha, Herrn Stengel, Herrn Merwald, Herrn Fischer, Frau Dr. Schulz-Kühnel, Herrn Ott und Herrn Mainberger einzulesen, wenn diese Daten vorher in der Datei stehen. Bei großen Dateien führt das zu ziemlich langen Wartezeiten. Aus diesem Grund bietet AmigaBASIC eine zweite Möglichkeit, Dateien zu organisieren: Die relativen Dateien. "Relativ" heißen sie, weil die einzelnen Datensätze "relativ" zum ersten Eintrag der Datei gelesen werden. Verständlicher wird das Ganze in der deutschen Übersetzung: "Dateien mit wahlfreiem Zugriff" oder "Direktzugriffs-Dateien".

Diese Namen lassen schon vermuten, was der Hauptunterschied zu sequentiellen Dateien ist: Sie können direkt auf jeden Datensatz zugreifen. "Datensatz" nennt man eine Ansammlung von zusammengehörigen Einzeldaten. Name, Straße, Wohnort und Telefonnumer sind Einzeldaten. Zusammen bilden sie einen Datensatz, nämlich eine Adresse. Für jede Person, die in Ihrer Adressenliste steht, gibt es einen Datensatz. Einen für Dr. Pleticha, einen für Herrn Stengel,... naja und so weiter eben. Bei relativen Dateien hat jeder Datensatz eine Nummer. Sie geben beim Lesen oder Schreiben an, welchen Satz Sie benutzen möchten. Das ist eigentlich alles. An welcher Stelle der gewünschte Datensatz auf der Diskette steht, und wie groß die Gesamtdatei ist, ist das Problem von AmigaDOS. Darum brauchen Sie sich nicht zu kümmern.

Obwohl dieses Prinzip nicht viel schwerer ist als das der sequentiellen Dateien, braucht man für die Programmierung einer relativer Datei ein paar Befehle mehr, als für eine vergleichbare sequentielle Datei. Aber keine Sorge, schwierig sind sie nicht.

Fangen wir ganz bescheiden an. Eine kleine Adressenverwaltung soll uns als Beispiel dienen. Auf unserer beiliegenden Diskette ist das die Datei "RelAdresse.schreiben".

```
OPEN "r",#1,"Adreßdatei.rel",92
FIELD #1,30 AS Nam$,30 AS Strasse$,20 AS Ort$,12 AS TelNR$

Eingabe:
  PRINT
  INPUT "Name:",EingabeNam$
  INPUT "Straße:",EingabeStrasse$
  INPUT "PLZOrt:",EingabeOrt$
  INPUT "Telefon:",EingabeTel$
  LSET Nam$=EingabeNam$
  LSET Strasse$=EingabeStrasse$
  LSET Ort$=EingabeOrt$
  LSET TelNR$=EingabeTel$
  x=x+1
  PUT #1,x
  PRINT "Adresse Nummer";x;": "Nam$
  INPUT "Weitere Adressen (J/N)";Antw$
  IF UCASE$(Antw$)<>"N" THEN Eingabe
CLOSE 1
```

Speichern Sie dieses kleine Programm bitte zunächst ab, falls Sie es selbst eingegeben haben. Danach schauen wir es uns etwas genauer an:

Für den OPEN-Befehl haben wir die Schreibweise verwendet, die Ihnen wahrscheinlich bisher weniger bekannt ist. OPEN hat ja zwei Syntax-Varianten, wie Sie sicher noch wissen. Für relative Dateien muß die zweite Schreibweise verwendet werden. Es gibt nämlich keinen Modus (wie bei FOR INPUT, FOR OUTPUT oder FOR APPEND), den Sie diesmal angeben könnten. Das kleine "r" steht für "Relativ". Dann folgt die Dateinummer und der Name der Datei. Der letzte Parameter gibt bei relativen Dateien die Satzlänge in Bytes an. Die Zahl in unserem Beispielprogramm besagt also, daß alle Einzeldaten der Adresse zusammen genau 92 Bytes lang sind. Bei sequentiellen Dateien können Sie an dieser Stelle die Länge des Dateipuffers angeben. Nichts anderes tun Sie auch hier: Für jede relative Datei legt AmigaBASIC einen Puffer an, der genau die Größe eines Datensatzes hat.

Der Befehl FIELD ist der erste wirklich neue Befehl. Mit ihm teilen Sie den Datensatz-Puffer in einzelne Bereiche ("Felder", engl.: "Fields") ein. Jedem dieser Bereiche wird dann eine Übergabe-Variable zugewiesen.

FIELD (Dateinummer),(Anzahl Bytes) AS (Variablenname), ...

In unserem Beispiel gibt's da zunächst die Variable 'Nam$', für die 30 Bytes zur Verfügung stehen. Der Inhalt von 'Nam$' darf pro Satz in der Datei nicht mehr als 30 Bytes benötigen. Er darf aber kürzer sein, doch dazu kommen wir etwas später beim LSET-Befehl. Mit den Werten im FIELD-Befehl legen Sie also die maximale Länge einer Variablen aus der Datei fest.
Für die weiteren Variablen läuft es ganz genauso: 'Strasse$' darf bis zu 30 Zeichen lang werden, 'Ort$' bis zu 20 und 'TelNR$' bis zu 12 Zeichen. Längere Adreßdaten werden abgeschnitten.

Im darauffolgenden 'Eingabe:'-Teil lesen wir die Adresse ein. Hier müssen wir für die Eingaben andere Variablen verwenden. Wir benutzen in unserem Programm 'EingabeNam$', 'EingabeStrasse$', 'EingabeOrt$' und 'EingabeTel$'. Sie können natürlich auch völlig andere Namen wählen. Einzige Bedingung: Es müssen andere Namen als in der FIELD-Zeile sein.
Warum? - Damit der nächste Befehl funktionieren kann: LSET.

Die Variablen aus FIELD sind Übergabevariablen. 'Nam$' ist der Name für einen 30 Bytes großen Bereich im Datei-Puffer. Der LSET-Befehl weist diesem Bereich einen Inhalt zu. In unserem Fall den Inhalt der Variablen 'EingabeNam$'. Dem Bereich mit mit dem Namen 'Strasse$' wird der Inhalt der Variablen 'EingabeStrasse$' zugewiesen. Und bei 'Ort$' und 'TelNR$' funktioniert's auch nicht anders. Wenn die zugewiesene Variable länger ist, als der zugehörige Pufferbereich, wird sie einfach abgeschnitten. Ist sie kürzer, bleibt der Rest des Bereichs ungenutzt. Die Variablen aus dem FIELD-Befehl haben also eine Sonderstellung: Ihnen wird mit LSET ein Inhalt zugewiesen, der dadurch im Datensatz-Puffer landet. Wenn Sie im Programm den

Inhalt einer solchen Übergabevariablen abfragen, erhalten Sie den Inhalt des Puffers an der durch FIELD definierten Stelle. Sie sehen das z.B. in der Zeile

```
PRINT "Adresse Nummer";x;": "Nam$
```

'Nam$' beinhaltet die ersten 30 Bytes des Pufferinhalts. In dem Augenblick, wo Sie 'Nam$' einen Wert zuweisen, ohne dafür den LSET-Befehl zu verwenden, (also z.B. durch INPUT oder mit 'Nam$=...') wird die Zuordnung zum Puffer aufgehoben. Die 30 Puffer-Bytes sind dann nicht mehr verfügbar. Deshalb bei der Dateieingabe der Umweg über die Variable 'EingabeNam$'.

Das Ganze gilt auch für die anderen Übergabevariablen des Datensatz-Puffers. Die Syntax von LSET ist ganz einfach:

LSET (Übergabevariable)=(Variable)

Kommen wir zum nächsten Schritt: Noch stehen die Daten ausschließich im Pufferspeicher. Um sie auf die Diskette zu bringen, benötigen Sie den PUT-Befehl. Der heißt zwar genauso, wie unser kleiner Grafiker aus Kapitel 4.1, aber die beiden sind nur sehr weitläufige Verwandte. Das ist fast wie bei Dallas, Denver oder Falcon Crest: Beim PUT-Befehl muß man ganz genau den PUT aus der Grafik-Linie der Familie und den aus der Daten-Linie unterscheiden. Falsch ist das Gerücht, daß der Grafik-PUT als brotloser Künstler das schwarze Schaf der Familie ist...

Hinter dem Daten-PUT geben Sie die Dateinummer an. Und dahinter die Nummer des Datensatzes, in den die aktuellen Pufferdaten geschrieben werden sollen. Beim Schreiben empfiehlt es sich, die Satznummern ab 1 hochzuzählen, so bleibt kein Satz ungenutzt. Sie können aber theoretisch jede beliebige Satznummer zwischen 1 und 16777215 angeben. Halt, Halt! Bevor Sie jetzt die gerade genannte Zahl ausprobieren, ein kleine Einschränkung: 16777215 ist die höchstmögliche Datensatznummer, die AmigaBASIC erlaubt. Welche Nummer in der Praxis die höchstmögliche ist, hängt vom zur Verfügung stehenden Speicherplatz und von der Datensatz-Größe ab. Auf jeden

Fall liegt sie deutlich unter 16777215 Sätzen, denn mit einer so großen Datei könnten Sie problemlos eine 20 Megabyte Harddisk bis aufs letzte Byte füllen. Jede gewöhnliche Amiga-Diskette geht da schon weit vorher aus den Nähten... Auf eine völlig leere 3½-Zoll-Diskette passen etwa 10000 Sätze unserer Adreßdatei.

Damit hätten wir's, soweit das Schreiben betroffen ist. Der Rest des Programms bietet Ihnen nichts Neues mehr. Speichern Sie doch bitte mal so um die 10 Adressen ab, damit das nächste Beispielprogramm ein bißchen was zu lesen hat. Sie finden es auf unserer Diskette im Buch unter dem Namen "RelAdresse.lesen".

```
OPEN "r",#1,"Adreßdatei",92
FIELD #1,30 AS Nam$,30 AS Strasse$,20 AS Ort$,12 AS TelNR$

Einlesen:
  INPUT "Adresse Nr.:";Nr
  GET #1,Nr
  IF EOF(1) THEN PRINT "Nummer nicht erlaubt!" : GOTO Einlesen
  PRINT Nam$
  PRINT Strasse$
  PRINT Ort$
  PRINT TelNR$
  INPUT "Weitere Adressen lesen (J/N)";Antw$
  IF UCASE$(Antw$)<>"N" THEN Einlesen
CLOSE 1
```

Die Programmzeilen mit dem OPEN- und dem FIELD-Befehl unterscheiden sich nicht vom Schreib-Programm: Die Datei wird geöffnet und der zugehörige Puffer eingerichtet. Die Puffereinteilung und die Satzlänge beim Lesen müssen identisch sein mit den Werten, die beim Schreiben der Datei verwendet wurden. In größeren Datei-Programmen gibt es deshalb oft zusätzlich zur relativen oft eine sequentielle Datei, aus der das Programm die notwendigen Informationen über Satzlänge und Feld-Einteilung liest.

Da Sie beim Öffnen relativer Dateien keinen Lese- oder Schreib-Modus angeben, können Sie für ein- und dieselbe Datei in beliebiger Reihenfolge abwechselnd Lese- und Schreibbefehle durchführen. Auch das ist ein Vorteil im Vergleich zu sequentiellen Dateien.

In unserem Adressen-Leseprogramm werden Sie zuerst nach der Nummer der Adresse gefragt, also der Satznummer. Wenn Sie in Ihre Datei 10 Adressen geschrieben haben, können Sie Zahlen zwischen 1 und 10 angeben. Bei weniger Adressen liegt die Grenze entsprechend tiefer. Jetzt betritt der Partner des PUT-Befehls die Bühne: GET. Auch von diesem Befehl gibt es eine Grafik- und eine Dateiversion (die Familienschicksale gleichen sich...). Der Datei-GET-Befehl liest einen angegebenen Datensatz von Diskette und schreibt die Daten in den Puffer. Sie müssen nur die Dateinummer und die gewünschte Satznummer angeben. Nachdem der GET-Befehl den Satz von der Diskette in den Puffer übertragen hat, können Sie den Pufferinhalt aus den Übergabevariablen lesen. So kommen wir dann an unsere Adreßdaten.

Die Funktion EOF funktioniert übrigens bei relativen Dateien genauso wie bei sequentiellen: Wenn Sie eine zu hohe Satznummer lesen wollen, ergibt EOF(1) den Wert -1. So läßt sich feststellen, ob der gewünschte Satz überhaupt existiert. Wenn sich Ihr Programm nicht um diese Abfrage kümmert, gelangen recht seltsame Daten in den Puffer: entweder Fragmente und zufällige Mischungen vorhandener Datensätze oder Anhäufungen von zufälligen Zeichen, meist Sonderzeichen. Lese-Programme sollten so ein Daten-Chaos nach Möglichkeit verhindern.

Die PRINT-Befehle bringen dann die Adresse auf den Bildschirm. Wie schon erklärt, können Sie die Einzeldaten nach dem GET-Befehl jederzeit aus den Übergabevariablen lesen: 'Nam$', 'Strasse$', 'Ort$' und 'TelNR$' liefern die zugewiesenen Pufferinhalte.

Unser Programm fragt dann noch, ob weitere Daten eingelesen werden sollen oder nicht. Wenn Sie "N" eingeben, wird die Datei geschlossen und das Programm beendet. Bei jeder anderen Eingabe können Sie einen neuen Datensatz lesen.

Beim Ausprobieren erleben Sie dann selbst die Vorteile relativer Dateien. Lesen Sie Adresse Nummer 1, Adresse 10, dann vielleicht Adresse 2 und danach Adresse 8: Die Daten sind ohne langes Suchen sofort verfügbar.

Damit Sie sich etwas ausführlicher von diesen Vorteilen überzeugen können, dabei auch noch etwas Neues lernen und zum Schluß ein nützliches Programm besitzen, machen wir uns jetzt mit unseren Fachkenntnissen auf dem Gebiet der relativen Dateien ans Programmieren eines neuen Utilities. Was dem Bundesnachrichtendienst, den Punktesammlern in Flensburg oder dem Finanzamt recht ist, sollte uns nur billig sein: Wir programmieren uns eine eigene kleine Datenbank.

5.2 Mick Jagger, Beethoven oder Wiener Würstchen - das Datenbank-Programm

Eigentlich haben wir schon eine Datenbank zusammen programmiert: Die Statistikdaten-Verwaltung ist schließlich in erster Linie auch nur zum Sammeln von Daten gedacht. Da es aber wenig sinnvoll ist, aus Ihren Adressen oder Ihren Schallplattenbeständen eine Balken- oder eine Tortengrafik zu zeichnen und außerdem dem Statistik-Programm die Möglichkeit fehlt, mehr als zwei Daten (nämlich Bezeichnung und Wert) pro Datensatz zu verwalten, haben wir uns entschlossen, mit Ihnen ein kleines offenes Datenbank-Programm zu erstellen. "Offen" nennen wir das Programm deshalb, weil Sie die Möglichkeit haben, eine Datei vollständig nach Ihren eigenen Wünschen einzurichten. Das heißt, mit dieser Datenbank kann Vati später seine klassischen Platten archivieren, Sohnemann seine Popcassetten und CDs und Mutti ihre Küchenrezepte. Und jeder so, wie er es gerne möchte.

Doch nun genug der Vorrede. Stürzen wir uns auf unser nächstes Projekt (Auch diesmal sollten Sie sich vielleicht mit Erdnüssen und Getränken eindecken...). Wer's einfach liebt, lädt natürlich wieder das Programm von unserer Diskette. "Datenbank" heißt das gute Stück.

```
Vorbereitungen:
  PALETTE 0,0,.1,.4
  PALETTE 2,0,1,0

Anfang:
  CLS : LOCATE 1,1 : PRINT "Auswahl"
  LOCATE 1,25 : COLOR 3,0 : PRINT "Datei:"; : COLOR 1,0
  IF Altname$<>"" THEN PRINT Altname$ ELSE PRINT "(keine)"
  PRINT
  COLOR 0,3 : PRINT SPACE$(21)"AmigaBASIC Datenbank"SPACE$(21)
  LOCATE 5,22 : COLOR 3,0
  PRINT "Bitte wählen Sie:"
  LOCATE 7,22
  COLOR 0,1 : PRINT " 1 "; : COLOR 1,0 : PRINT " Datei einrichten"
  LOCATE 9,22
  COLOR 0,1 : PRINT " 2 "; : COLOR 1,0 : PRINT " Neue Daten eingeben"
  LOCATE 11,22
  COLOR 0,1 : PRINT " 3 "; : COLOR 1,0 : PRINT " Daten lesen"
  LOCATE 13,22
  COLOR 0,1 : PRINT " 4 "; : COLOR 1,0 : PRINT " Daten suchen"
  LOCATE 15,22
  COLOR 0,1 : PRINT " 5 "; : COLOR 1,0 : PRINT " Beenden"
Auswahl:
  LOCATE 18,1 : PRINT SPACE$(60)
  LOCATE 18,22 : COLOR 3,0 : PRINT "Ihre Wahl:";
  COLOR 1,0 : LINE INPUT Wahl$
  Wahl$=LEFT$(Wahl$,1)
  IF Wahl$<"1" OR Wahl$>"5" THEN Auswahl
  IF Wahl$="1" THEN DateiEinrichten
  IF Wahl$="2" THEN DateiEingeben
  IF Wahl$="3" THEN DatSuch=0 : GOTO DatenLesen
  IF Wahl$="4" THEN DatSuch=1 : GOTO DatenLesen
  PRINT "Programm beendet."
END
```

An 'Vorbereitungen:' gibt's eigentlich wenig: Zwei kleine PALETTE-Befehle haben wir anzubieten und selbst die nur aus kosmetischen Gründen. Wir machen damit das Blau des Bildschirmhintergrunds etwas dunkler. Das gibt dem Programm gleich ein professionelleres Aussehen. Lachen Sie nicht! Es stimmt wirklich: Der optische Eindruck eines Programms verbessert sich, wenn Sie nicht die Standard-Workbench-Farben verwenden. Das Ganze sieht dann nicht mehr so nach BASIC aus. Mit dem zweiten PALETTE ändern wir das Workbench-Schwarz in Grün um. Diese Farbe bietet einen besseren Kontrast zum Hintergrund.

Der Programmteil 'Anfang:' löscht den Bildschirm und schreibt in die linke obere Bildschirmecke das Wort "Auswahl". In unserem Datenbank-Programm schreiben wir an diese Stelle immer den aktuellen Modus. So sieht der Anwender gleich, wo und woran er ist.

In der Mitte der ersten Zeile erscheint die aktuelle Datei. Unmittelbar nach dem Programmstart wurde natürlich noch keine Datei benutzt, deshalb erscheint hier das Wort "(keine)".

Dann drucken wir eine Überschriftsleiste. Hier kommt ein neuer Befehl vor: SPACE$(x). "Space" (deutsch: "Raum") ist die englische Bezeichnung für die Leertaste auf der Tastatur. SPACE$ erzeugt einen String aus lauter Leerzeichen. In Klammern geben Sie die gewünschte Länge an. So können Sie (wie in der Überschriftszeile) farbige Balken drucken: Einfach eine Schriftfarbe zur Hintergrundfarbe machen, wie zum Beispiel durch COLOR 0,3. In der Überschriftsleiste schreiben wir dunkelblau auf orange. SPACE$ eignet sich auch zum Löschen von Texten, aber dafür kommen später noch Beispiele.

Als nächstes drucken wir ein Auswahlmenü, das die einzelnen möglichen Punkte vorstellt. "Datei einrichten" rufen Sie auf, um eine neue Datei zu erstellen. In diesem Programmteil werden die Namen und Längen der einzelnen Datenfelder festgelegt. "Neue Daten eingeben" ist der Programmteil, in dem Sie Daten eingeben können. "Daten lesen" ermöglicht Ihnen, durch eine ganze Datei zu blättern und einzelne Datensätze zu verändern. "Daten

suchen" hilft Ihnen dabei, Datensätze nach bestimmten Kriterien zu suchen. "Beenden" erklärt sich wirklich von selbst: Mit dieser Option macht Ihr Programm Schluß. Es folgen die Programmzeilen, die für die Auswahl zuständig sind. Hier sehen Sie auch einen SPACE$ zu Löschzwecken: Was immer der Anwender in der Auswahl-Zeile getippt hat - wenn es eine Fehleingabe war, wird sie gelöscht und eine neue Abfrage findet statt.

Abhängig von der Eingabe werden die einzelnen Programmteile aufgerufen. Für "Daten lesen" und "Daten suchen" wird in beiden Fällen 'DatenLesen:' verwendet. Anhand der Variablen 'DatSuch' erkennt das Programm, ob es alle Datensätze einlesen soll oder nur die gesuchten. Und schon kommt der erste Programmteil:

```
DateiEinrichten
  CLS : LOCATE 1,1 : COLOR 1,0 : PRINT "Datei einrichten"
  LOCATE 1,25 : COLOR 3,0 : PRINT "Datei";
  COLOR 1,0 : PRINT "(keine)"
  COLOR 3,0 : LOCATE 3,1
  PRINT "Bitte geben Sie die Feldnamen und die Feldlängen an."
  COLOR 1,0
  FOR x=0 TO 9
    Feldname$="" : Laenge(x)=0
  NEXT x
  LOCATE 4,1 : PRINT "Name" : LOCATE 4,26 : PRINT "Länge (-40)"
  FOR x=0 TO 9
    AnzahlFelder=x
    LOCATE x+6,1 : LINE INPUT Feldname$(x)
    IF Feldname$(x)="" THEN x=10 : AnzahlFelder=AnzahlFelder-1
    Feldname$(x)=LEFT$(Feldname$(x),25)
    LOCATE x+6,26 : PRINT SPACE$(40);
    LOCATE x+6,26 : LINE INPUT Laenge$
    IF Laenge$="" OR ABS(VAL(Laenge$)) > 40 THEN Laenge$="40"
    Laenge(x)=INT(ABS(VAL(Laenge$)))
    IF Laenge(x)=0 THEN Laenge(x)=40
  NEXT x
```

```
Fehlerabfrage:
  GOSUB FehlerJN
  IF Korr=0 THEN DateiOeffnen
  IF Korr=1 THEN Fehlerkorrektur
GOTO Fehlerabfrage

Fehlerkorrektur:
  FOR x=0 TO AnzahlFelder
    LOCATE x+6,1 : PRINT SPACE$(60)
    LOCATE x+6,25 : PRINT Laenge(x)
    LOCATE x+6,1 : PRINT Feldname$(x)
  NEXT x
  FOR x=0 TO AnzahlFelder
    LOCATE x+6,1 : LINE INPUT Feldname$
    IF Feldname$<>"" THEN Feldname$(x)=LEFT$(Feldname$,25)
    LOCATE x+6,26 : LINE INPUT Laenge$
    IF ABS(VAL(Laenge$))>40 THEN Laenge$="40"
    IF Laenge$<>"" THEN Laenge(x)=INT(ABS(VAL(Laenge$)))
    IF Laenge(x)=0 THEN Laenge(x)=40
  NEXT x
GOTO Fehlerabfrage

DateiOeffnen:
  LOCATE 19,1 : PRINT SPACE$(60);
  LOCATE 19,1 : COLOR 3,0 : PRINT "Bitte geben Sie den Dateinamen ein:";
  COLOR 1,0 : LINE INPUT Nam$
  Satzlaenge=0
  FOR x=0 TO AnzahlFelder
    Satzlaenge=Satzlaenge+Laenge(x)
  NEXT x
  IF Nam$="" OR Satzlaenge=0 THEN BEEP : GOTO Anfang
  OPEN "R",#1,Nam$,Satzlaenge
    FIELD #1,Laenge(0) AS Dat$(0),Laenge(1) AS Dat$(1),Laenge(2) AS Dat$
(2),Laenge(3) AS Dat$(3),Laenge(4) AS Dat$(4),Laenge(5) AS Dat$(5),Laeng
e(6) AS Dat$(6),Laenge(7) AS Dat$(7),Laenge(8) AS Dat$(8),Laenge(9) AS D
at$(9)
```

Also, an dieser Stelle müssen wir doch mal unterbrechen. Das gerade war wahrscheinlich die längste Zeile, die Sie je in AmigaBASIC eingegeben haben. Zumindest dann, wenn Sie sich entschlossen haben, unser Programm einzutippen, statt es von Diskette zu laden.

Programmzeilen dürfen ja bis zu 255 Zeichen lang sein, aber für guten Programmierstil spricht das nicht gerade. In diesem Fall ließ es sich allerdings leider nicht vermeiden. Sämtliche Bereichsdefinitionen mit FIELD müssen in einer Zeile stehen. Beginnt nämlich eine neue FIELD-Zeile, überschreibt sie die alte Puffereinteilung. Da wir 10 Variablen unterbrigen wollen, mußten wir also diese Riesenzeile konstruieren. Tippen Sie einfach alles hintereinander weg, und drücken Sie am Schluß <RETURN>.

Dann geht's normal weiter:

```
    FOR x=1 TO AnzahlFelder
      LSET Dat$(x)=" "
    NEXT x
  CLOSE 1
  OPEN Nam$+".Felder" FOR OUTPUT AS 2
    PRINT #2,AnzahlFelder
    PRINT #2,Satzlaenge
    PRINT #2,0
    FOR x=0 TO AnzahlFelder
      WRITE #2,Feldname$(x)
      PRINT #2,Laenge(x)
    NEXT x
  CLOSE 2
  Altname$=Nam$
GOTO Anfang
```

In diesem Teil richten Sie sich also gemütlich ein in Ihrer Datei. Zuerst drucken wir den aktuellen Modus ins linke obere Eck: "Datei einrichten". Die aktuell bearbeitete Datei heißt "(keine)". Schließlich basteln wir gerade an einer neuen Datei, und die hat noch keinen Namen.

Übrigens: Vielleicht wundern Sie sich über den inflationären Gebrauch von COLOR-Befehlen im Listing. Wir haben versucht, durch die Schriftfarben eine optische Gliederung zu erzeugen: Hinweise, Aufforderungen und Erklärungen schreiben wir orange. Die Daten, Übersichten und Anwendereingaben erscheinen weiß auf dem Bildschirm. Die Feldnamen in den Datensätzen schließlich erscheinen grün. Mit COLOR 3,0 schalten wir auf Orange, mit COLOR 2,0 auf Grün und mit COLOR 1,0 auf Weiß.

Im Teil 'DateiEinrichten:' wird der Anwender aufgefordert, die Namen und Längen der Datenfelder festzulegen. Diese Einteilung können Sie so vornehmen, wie es Ihnen sinnvoll erscheint. Jeder Datensatz kann aus bis zu 10 Einzelelementen bestehen. Deshalb die Schleife FOR x=0 TO 9.

In der Variablen 'AnzahlFelder' merkt sich das Programm die Anzahl der Datenfelder pro Datensatz. Mit jeder neuen Feldeingabe wird dieser Wert um eins erhöht.

Zuerst lesen wir den Feldnamen ein. Das sind die Bezeichnungen wie "Name", "Straße", "Ort", "Titel" etc. Die Maximallänge für einen Feldnamen sind 25 Zeichen. Schließlich brauchen wir noch ein bißchen Platz für die eigentlichen Daten. Findet in diesem Modus eine Leereingabe statt, erkennt das Programm daraus, daß nun alle Felder eingegeben sind. 'x' wird auf 10 gesetzt, die Schleife wird also beim nächsten NEXT beendet.

Ist der Feldname eingegeben, fehlt noch die Feldlänge. Maximalwert für die Datenfeldlänge sind 40 Zeichen. Zunächst löscht ein SPACE$(40) eventuell überstehende Teile des Feldnamens. Der wird ja nach 25 Zeichen abgeschnitten. Nun gibt der Anwender einen Wert für die Variable 'Laenge$' ein. Gab es eine Leereingabe oder eine Eingabe größer als 40, wird der Wert für die Feldlänge auf 40 Zeichen begrenzt. Mit der reichlich verschachtelten Funkion INT(ABS(VAL(Laenge$))) filtern wir so ziemlich alle Fehler aus der Eingabe, die der Anwender bewußt oder unbewußt gemacht hat: VAL wandelt den String in eine Zahl. Sind Buchstaben oder Sonderzeichen dabei, fallen sie unter den Tisch. ABS macht die Zahl in jedem Fall positiv: Eingaben

mit einem Minuszeichen werden "umgepolt". INT schließlich entledigt uns eines eventuell vorhandenen Kommateils. Der würde bei Feldlängen ja wirklich keinen Sinn machen. Oder was verstehen Sie unter 23,124246 Zeichen? Sie sehen unser Sicherheitscheck ist fast so gut wie der an Flughäfen. Hat der Anwender 0 eingegeben, wird ebenfalls der Standardwert 40 übernommen. Wir hoffen, daß wir an alle möglichen und unmöglichen Fehler gedacht haben. Wenn Sie eine persönliche mögliche Fehleingabe entdecken, rüsten Sie Ihr Programm halt einfach entsprechend nach.

Die Schleife läuft höchstens 10 mal durch. Dann gibt es 10 Feldnamen, der höchste erlaubte Wert.

Im Programmteil 'Fehlerabfrage:' wird ein Unterprogramm namens 'FehlerJN:' (Fehler Ja/Nein) aufgerufen, daß es bisher noch gar nicht gibt. Es wird den Anwender später fragen, ob er bei seiner Eingabe einen Fehler entdeckt hat. Antwortet er mit Ja, erhält er Gelegenheit, seinen Fehler zu korrigieren. Die Variable 'Korr' gibt Auskunft über die Antwort. Besteht der Wunsch nach Fehlerkorrektur, wird der gleichnamige Programmteil aufgerufen: 'Fehlerkorrektur:'. Hier drucken wir zunächst die bisher eingegebenen Daten auf den Bildschirm: Spaltenweise untereinander die Feldnamen und Feldlängen. Eine FOR...NEXT-Schleife zählt von 1 bis zu 'AnzahlFelder': Für jedes eingegebene Feld gibt es eine Korrekturmöglichkeit. Feld für Feld erscheint der Cursor und ein LINE INPUT fragt nach neuen Eingaben. Ist die Antwort ein Leerstring, soll nichts an den bisherigen Daten verändert werden. Bei anderen Eingaben wird der neue Wert anstelle des alten übernommen. Die Abfragen zur Einschränkung der Feldlänge sind identisch mit den Programmzeilen, die im 'DateiEinrichten:"-Teil dieselbe Funktion erfüllen.

Nach erfolgter Korrektur wird das Label 'Fehlerabfrage:' erneut angesprungen, falls Sie Verbesserungen an Ihren Verbesserungen vornehmen wollen. Das Ganze läuft so lange im Kreis, bis Sie erklären, daß Sie keine Korrekturen mehr wünschen.

Dann kommt der Teil 'DateiOeffnen:' dran: Er fragt zunächst nach einem Namen, den die angelegte Datei erhalten soll. Eine FOR...NEXT-Schleife errechnet aus den Feldinhalten von 'Laenge(x)' die tatsächliche 'Satzlaenge'. Ist diese 'Satzlaenge'=0, wurden also keine Felder für den Datensatz definiert, kehren wir mit einem Protest-BEEP zum 'Anfang:' zurück. Anderenfalls wird die relative Datei mit dem in 'Nam$' angegebenen Namen eröffnet. Jetzt kommt noch die vorher schon angesprochene FIELD-Zeile. Tippen Sie sie bitte sehr sorgfältig in eine einzige Zeile ab.

Wie Sie sehen, können Sie auch Feldvariablen wie 'Dat$(x)' als Übergabevariablen benutzen. Da wir sowohl die Anzahl der Daten pro Satz, als auch deren Namen und deren Länge variabel halten wollen, haben wir uns für diese Lösung entschieden. Jedes Element des Datenfelds 'Dat$(x)' bekommt im Puffer einen Bereich mit der Länge 'Laenge(x)'. Da diese Länge bei nicht benutzten Feldelementen Null ist, geht durch die überzähligen Variablen trotzdem kein Pufferspeicher verloren.

Weil wir halt nun mal 10 Zuweisungen in einer FIELD-Anweisung unterbringen müssen, wurde die ganze Sache leider etwas unübersichtlich.

Anschließend werden die Datenfelder im Puffer mit Leerzeichen aufgefüllt. So löschen wir eventuell vorhandene Daten. Damit sind alle Vorbereitungen der Relativ-Datei abgeschlossen.

Wie wir im letzten Kapitel schon erwähnten, bietet es sich an, zusätzlich eine sequentielle Datei anzulegen, in der wir die 'AnzahlFelder', die 'Satzlänge', die Anzahl an Datensätzen (zur Zeit 0), die Feldnamen und -längen abspeichern. Die sequentielle Datei bekommt den Namen der relativen mit dem Anhängsel ".Felder". Die Datei zu "Adressen" hieße also "Adressen.Felder". Ist auch diese Datei geschrieben, merkt sich 'Altname$' noch den letzten benutzen Dateinamen, und das Programm springt zurück zum Anfang.

Nun, da wir schon eine so schöne Relativ-Datei haben, sollten wir sie auch bald mit Inhalten füllen:

```
DatenEingeben:
  CLS : LOCATE 1,1 : PRINT "Neue Daten eingeben"
  IF Nam$="" THEN
    LOCATE 3,1 : COLOR 3,0 : PRINT "Bitte geben Sie den Dateinamen ein:"
    COLOR 1,0 : LINE INPUT Nam$
    IF Nam$= "=" OR Nam$="*" THEN Nam$=Altname$
    IF Nam$="" THEN Anfang
  END IF
  GOSUB Dat.FelderExistJN
  IF DateiExist=0 THEN
    COLOR 3,0 : PRINT
    PRINT "Bitte drücken Sie eine Taste."
    WHILE INKEY$="" : WEND : COLOR 1,0
    GOTO Anfang
  END IF
  GOSUB Dat.FelderLesen
  SatzNummer=AnzahlSaetze+1

  OPEN "R",#1,Nam$,Satzlaenge
    FIELD #1,Laenge(0) AS Dat$(0),Laenge(1) AS Dat$(1),Laenge(2) AS Dat$
(2),Laenge(3) AS Dat$(3),Laenge(4) AS Dat$(4),Laenge(5) AS Dat$(5),Laeng
e(6) AS Dat$(6),Laenge(7) AS Dat$(7),Laenge(8) AS Dat$(8),Laenge(9) AS D
at$(9)
```

Schon wieder dieses Schlacht-FIELD. Am besten, Sie benutzen zur Eingabe "Copy" & "Paste" und kopieren die Zeile von oben an diese Stelle. Und jetzt lassen Sie sich nicht weiter stören...

```
Eingabeschleife:
      CLS : LOCATE 1,1 : COLOR 1,0 : PRINT "Neue Daten eingeben"
      LOCATE 1,25 : COLOR 3,0 : PRINT "Datei:";
      COLOR 1,0 : PRINT Nam$

      Eing=0
      LOCATE 1,50 : PRINT "Satz:";SatzNummer
      PRINT : COLOR 3,0
      PRINT "Bitte geben Sie die neuen Daten ein:" : COLOR 1,0
```

```
    FOR x=0 TO AnzahlFelder
      LOCATE 5+x,1 : COLOR 2,0 : PRINT Feldname$(x)": "
    NEXT x : COLOR 1,0
    FOR x=0 TO AnzahlFelder
      LOCATE 5+x,LEN(Feldname$(x))+2
      LINE INPUT Eingabe$
      IF Eingabe$<>"" THEN Eing=1
      Eingabe$(x)=LEFT$(Eingabe$,Laenge(x))
      LSET Dat$(x)=Eingabe$(x)
    NEXT x
Fehlerabfrage2:
    GOSUB FehlerJN
    IF Korr=0 THEN SatzSchreiben
    IF Korr=1 THEN EingabeKorr
  GOTO Fehlerabfrage2

EingabeKorr:
    CLS : LOCATE 1,1 : COLOR 1,0 : PRINT "Daten ändern"
    LOCATE 1,25 : COLOR 3,0 : PRINT "Datei:";
    COLOR 1,0 : PRINT Nam$

    LOCATE 1,50 : PRINT "Satz:";SatzNummer
    PRINT : PRINT
    FOR x=0 TO AnzahlFelder
      LOCATE 5+x,1 : COLOR 2,0 : PRINT Feldname$(x)": ";
      COLOR 1,0 : PRINT Eingabe$(x)
    NEXT x
    FOR x=0 TO AnzahlFelder
      LOCATE 5+x,LEN(Feldname$(x))+2
      LINE INPUT Eingabe$
      IF Eingabe$<>"" THEN
        Eing=1
        Eingabe$(x)=LEFT$(Eingabe$,Laenge(x))
        LSET Dat$(x)=Eingabe$(x)
      END IF
    NEXT x
  GOTO Fehlerabfrage2

SatzSchreiben:
  IF Eing=1 THEN
```

```
      PUT #1,SatzNummer
      IF DatAend=1 THEN DatAend=0 : GOTO Leseschleife
      SatzNummer=SatzNummer+1
    END IF
    IF DatAend=1 THEN DatAend=0 : GOTO Leseschleife

WeitereJN:
    LOCATE 19,1 : PRINT SPACE$(60) : COLOR 3,0
    LOCATE 19,1 : PRINT "Weitere Sätze ? (J/N)";
    COLOR 1,0 : LINE INPUT a$
    IF UCASE$(a$)="J" OR a$="" THEN Eingabeschleife
    IF UCATE$(a$)="N" THEN DateiSchliessen
GOTO WeitereJN

DateiSchliessen:
    CLOSE 1
    OPEN Nam$+".Felder" FOR OUTPUT AS 2
      PRINT #2,AnzahlFelder
      PRINT #2,Satzlaenge
      PRINT #2,SatzNummer-1
      FOR x=0 TO AnzahlFelder
        WRITE #2,Feldname$(x)
        PRINT #2,Laenge(x)
      NEXT x
    CLOSE 1
GOTO Anfang
```

Der Programmteil 'DatenEingeben:' hinterläßt zunächst seine Visitenkarte im linken oberen Bildschirmeck. Hat 'Nam$' keinen Inhalt, soll ein neuer Dateiname eingelesen werden. Dabei können Sie wie in unseren anderen Programmen durch ein = oder ein *-Zeichen den letzten verwendeten Namen abkürzen.

Hinter diesen Zeilen steckt folgendes: Normalerweise löschen die einzelnen Programmteile den 'Nam$', nachdem der Dateizugriff erfolgte. Nur 'DateiEinrichten:' tut das nicht: Wenn Sie direkt von der Datei-Vorbereitung in die Dateneingabe wechseln, geben Sie automatisch Daten in die gerade angelegte Datei ein.

Man kann normalerweise davon ausgehen, daß frisch eingerichtete Dateien auch mit Werten versehen werden sollen. Deshalb wurde diese Funktion automatisiert.

Als nächstes rufen wir das Unterprogramm 'Dat.FelderExistJN:' auf. Es stellt fest, ob zum eingegebenen Dateinamen eine .Felder-Datei exisitert. Das erklärt wohl auch den etwas unübersichtlichen Label-Namen. Dieses Unterprogramm liefert sein Ergebnis in der Variablen 'DateiExist'. Hat sie den Wert 0, gibt es die Datei nicht. Bei 1 wurde sie gefunden. Wenn die .Felder-Datei existiert, gibt es auch die zugehörige Relativ-Datei. Wir wollen feststellen, ob eine bestimmte Datei bereits existiert. Und zwar ohne einen "File not found"-Error zu erzeugen. Den Trick, wie wir das geschafft haben, werden Sie kennenlernen, wenn wir das Unterprogramm eingeben. Gibt es die Datei nicht, wird der Anwender darauf hingewiesen. Er muß zur Bestätigung eine beliebige Taste drücken, dann erfolgt der Rücksprung zum 'Anfang:'.

Das Unterprogramm 'Dat.FelderLesen:' liest die .Felder-Datei, die zur gewählten Datei gehört. So werden die Werte der Variablen wie 'AnzahlFelder', 'Satzlaenge' und 'AnzahlSaetze' beschafft, außerdem die Feldnamen und -längen. Da wir neue Daten grundsätzlich hinter dem letzten Satz anhängen wollen, ergibt sich die neue 'SatzNummer' aus 'AnzahlSaetze'+1. Mit den festgestellten Daten wird die Relativ-Datei eröffnet, und auch die FIELD-Zeile von vorhin treibt hier wieder ihr Unwesen.

Nun beginnt der eigentliche Eingabeteil: 'Eingabeschleife:'. Auch dieser Modus weist sich aus ("Neue Daten eingeben") und druckt den aktuellen Dateinamen in die Mitte der ersten Bildschirmzeile. Nach diesen Formalitäten wird eine Variable namens 'Eing' auf 0 gesetzt. Sie steht für "Eingabe" und soll später mitteilen, ob Eingaben stattfanden oder nicht.

In die rechte Bildschirmecke paßt gerade noch die aktuelle Satznummer, womit auch sie dem Anwender mitgeteilt wäre. Nun dürfen die neuen Daten eingegeben werden. Das Programm weist mit einer entsprechenden Aufforderung darauf hin. Von 0 bis zu 'AnzahlFelder' erscheinen nun die Feldnamen auf dem

Bildschirm. Der Eingabecursor wandert hinter den ersten Namen und wartet dort mit LINE INPUT auf eine Eingabe. Die erste nicht leere Eingabe setzt 'Eing' auf 1: Eine richtige Eingabe fand statt. Der eingegebene 'Eingabe$' wird gegebenenfalls auf die zugehörige Feldlänge gestutzt und dann mit LSET in den Dateipuffer geschrieben. Das Feld 'Eingabe$(x)' merkt sich außerdem noch mal alle Eingaben des aktuellen Datensatzes, unabhängig vom Dateipuffer.

Ist die Schleife durchlaufen, erhält der Anwender wieder Gelegenheit zur Fehlerkorrektur: 'FehlerJN:' fragt die Antwort ab und teilt sie in 'Korr' mit. Beim Wunsch nach Korrektur tritt 'EingabeKorr:' in Aktion. Dieser Programmteil bringt zuerst den Text 'Daten ändern' ins Bildschirmeck links oben. Auch Datei und Satznummer finden ihren Platz. Die nächste FOR...NEXT-Scheife druckt Feldnamen und Feldinhalte auf den Bildschirm. Es folgt die Eingabe-Schleife, über die es nichts Neues mehr zu berichten gibt. Sind alle Korrekturen erledigt, wird der Satz von 'SatzSchreiben:' mit dem PUT-Befehl auf Diskette geschrieben. Allerdings nur, falls auch Korrekturen stattfanden (falls also 'Eing'=1).

Ein Wort zu Variablen 'DatAend': Das 'DatenLesen:'-Programm leiht sich zum Ändern von Daten bei Bedarf die 'EingabeKorr:"-Routine aus. Wenn 'DatAend' den Wert 1 hat, soll nach Beendigung der Eingabe zur 'Leseschleife:' zurückgesprungen werden.

Ist dies nicht der Fall, wird die 'SatzNummer' um 1 erhöht und eine neue Eingabe kann stattfinden. Dazu fragt der Teil 'WeitereJN:' beim Anwender nach, ob weitere Sätze folgen. Wenn ja - auf zur Eingabe. Wenn nein, kann die Datei geschlossen werden. Hintendrein schreiben wir noch eine neue .Felder-Datei mit den aktuellen Werten. Beachten Sie, daß diesmal 'Nam$' gelöscht wird, bevor der Rücksprung zum 'Anfang:' durchgeführt wird. Vor dem nächsten Dateizugriff muß also erst ein neuer Dateiname eingeben werden.

```
DatenLesen:
    CLS : LOCATE 1,1 : PRINT "Daten lesen"
    IF DatSuch=1 THEN LOCATE 1,1 : PRINT "Daten suchen"
    LOCATE 3,1 : COLOR 3,0 : PRINT "Bitte geben Sie den Dateinamen ein:"
    COLOR 1,0 : LINE INPUT Nam$
    IF Nam$= "=" OR Nam$="*" THEN Nam$=Altname$
    IF Nam$="" THEN Anfang
    Altname$=Nam$

    GOSUB Dat.FelderExistJN
    IF DateiExist=0 THEN
      PRINT : COLOR 3,0
      PRINT "Bitte drücken Sie eine Taste."
      COLOR 1,0
      WHILE INKEY$="" : WEND
      GOTO Anfang
    END IF
    GOSUB Dat.FelderLesen
    IF AnzahlSaetze=0 THEN
      PRINT : BEEP
      COLOR 1,0
      PRINT "Die Datei enthält noch keine Sätze!"
      PRINT : COLOR 3,0
      PRINT "Bitte drücken Sie eine Taste."
      COLOR 1,0
      WHILE INKEY$="" : WEND
      GOTO Anfang
    END IF
  IF DatSuch=1 THEN GOSUB DatenSuchen

  OPEN "R",#1,Nam$,Satzlaenge
    FIELD #1,Laenge(0) AS Dat$(0),Laenge(1) AS Dat$(1),Laenge(2) AS Dat$
(2),Laenge(3) AS Dat$(3),Laenge(4) AS Dat$(4),Laenge(5) AS Dat$(5),Laeng
e(6) AS Dat$(6),Laenge(7) AS Dat$(7),Laenge(8) AS Dat$(8),Laenge(9) AS D
at$(9)
```

Na, Sie wissen ja schon...

```
    SatzNummer=1
Leseschleife:
    CLS : LOCATE 1,1 : COLOR 1,0 : PRINT "Daten lesen"
    LOCATE 1,25 : COLOR 3,0 : PRINT "Datei:";
    COLOR 1,0 : PRINT Nam$
    COLOR 3,0
    LOCATE 17,1 : PRINT "[Cursor nach oben] =Satz zurück"
    LOCATE 17,37 : PRINT "[CTRL]-[A]=erster Satz"
    PRINT "[Cursor nach unten]=Satz vor"
    LOCATE 18,37 : PRINT "[CTRL]-[E]=letzter Satz"
    PRINT "[CTRL]-[D]          =Satz drucken"
    LOCATE 19,37 : PRINT "[HELP]     =Satz ändern"
    PRINT "[CTRL]-[H]          =Hauptauswahl";
SatzLesen:
    COLOR 1,0
    IF SatzNummer>AnzahlSaetze THEN BEEP : SatzNummer=AnzahlSaetze
    IF SatzNummer<1 THEN BEEP : SatzNummer=1
    LOCATE 1,50 : PRINT "Satz:";SatzNummer
    GET #1,SatzNummer
    IF DatSuch=1 THEN LOCATE 1,1 : PRINT "Daten suchen": GOSUB SuchDaten
Pruefen
    IF DatSuch=1 AND Gefunden=0 THEN
      IF SatzNummer=AnzahlSaetze THEN Richtung=-1
      IF SatzNummer=AnzahlSaetze AND GefSatz=0 THEN
        CLS
        LOCATE 5,10 : PRINT "Keinen passenden Satz gefunden!"
        LOCATE 7,10 : COLOR 3,0
        PRINT "Bitte drücken Sie eine Taste."
        COLOR 1,0 : BEEP
        WHILE INKEY$="" : WEND : CLOSE 1 : GOTO Anfang
      END IF
      IF SatzNummer=1 THEN Richtung=1
      SatzNummer=SatzNummer+Richtung
      GOTO SatzLesen
    END IF
    GefSatz=1
    FOR x=0 TO AnzahlFelder
      LOCATE 5+x,1 : COLOR 2,0 : PRINT Feldname$(x)":"
```

```
    NEXT x : COLOR 1,0
    FOR x=0 TO AnzahlFelder
      LOCATE 5+x,LEN(Feldname$(x))+3
      PRINT Dat$(x)
      Eingabe$(x)=Dat$(x)
    NEXT x
    Tast$=""
    WHILE Tast$="" : Tast$=INKEY$ : WEND
    IF Tast$=CHR$(28) THEN SatzNummer=SatzNummer-1 : Richtung=-1
    IF Tast$=CHR$(29) THEN SatzNummer=SatzNummer+1 : Richtung=1
    IF Tast$=CHR$(1) THEN SatzNummer=1
    IF Tast$=CHR$(5) THEN SatzNummer=AnzahlSaetze
    IF Tast$=CHR$(8) THEN EndeLesen
    IF Tast$=CHR$(4) THEN
      FOR x=0 TO AnzahlFelder
        LPRINT Feldname$(x)":"Dat$(x)
      NEXT x
      LPRINT
    END IF
    IF Tast$=CHR$(139) THEN DatAend=1 : GOTO EingabeKorr
  GOTO Leseschleife

EndeLesen:
  CLOSE 1
  Nam$=""
GOTO Anfang
```

Ach übrigens: Wenn Sie tippen - haben Sie Ihr Programm heute schon gespeichert?

'DatenLesen:' beginnt wie gewohnt: Der Modus ("Daten lesen") wird an Position (1,1) geschrieben. Beim 'Auswahl:'-Teil haben Sie bereits gesehen, daß der Programmteil 'DatenLesen:' für zwei Auswahlpunkte benutzt wird: "Daten lesen" und "Daten suchen". Der Unterschied ist, daß im Such-Modus nur die Sätze auf dem Bildschirm erscheinen, die auf einen vorher eingebenen Such-Schlüssel passen. Die Variable 'DatSuch' sagt der Routine, in welchem Modus sie arbeitet. Hat 'DatSuch' den Wert 1, ist der

Such-Modus aktiv, ansonsten der Lese-Modus. Die erste Konsequenz von 'DatSuch'=1 ist, daß eine andere Bezeichnung ins Bildschimreck gedruckt wird: "Daten suchen".

Für beide Betriebsarten identisch ist der nächste Teil: Ein Dateiname wird eingegeben. Den letzten verwendeten Namen können Sie wieder mit = oder * abkürzen.

Das Unterprogramm 'Dat.FelderExistJN' wird aufgerufen, um festzustellen, ob eine Datei mit dem angegebenen Namen überhaupt auf der Diskette exitistiert. Falls nicht ('DateiExist'=0), druckt das Unterprogramm eine entsprechende Nachricht. Nach der Rückkehr vom Unterprogramm soll der Anwender dann eine Taste drücken, sobald er den Text gelesen hat. Wenn die Datei jedoch existiert, wird sie mit 'Dat.FelderLesen' gelesen.

Handelt es sich dabei um eine frisch angelegte Datei, kann es vorkommen, daß sich in der Datei noch keine Datensätze befinden. Dann gibt es natürlich auch nichts, was gelesen werden könnte. In diesem Fall bringt das Programm den Text "Die Datei enthält noch keine Sätze!" auf den Bildschirm, läßt diesen Hinweis wieder durch einen Tastendruck bestätigen und kehrt danach zum 'Anfang:' zurück. In die angegebene Datei müssen erst Daten geschrieben werden, dazu dient ja Menüpunkt Nummer 2.

Ist der Such-Modus aktiv ('DatSuch'=1), wird das Unterprogramm 'DatenSuchen' aufgerufen, wo der Anwender das Wort angibt, nach dem gesucht werden soll. Und jetzt endlich kann die Relativ-Datei eröffnet werden. Unsere FIELD-Anweisung kennen Sie ja mittlerweile zur Genüge. Beim Eintippen können Sie wieder die "Copy"- und die "Paste"-Option aus dem "Edit"-Pulldown einsetzen.

Beim Anzeigen der Daten beginnt das Programm grundsätzlich mit Satz 1. Also setzen wir zu Beginn 'SatzNummer' auf diesen Wert. Nun beginnt die 'Leseschleife:'. In diesem Teil werden die Daten gelesen und angezeigt. Zunächst weist sich der Programmteil wieder mit einer Modus-Bezeichnung aus, auch die aktuelle Datei wird angezeigt. Dann beginnt der Bildschirmaufbau. Unterhalb des Bereichs, wo die Datenfelder angezeigt wer-

den, gibt es für den Anwender eine Übersicht über die zur Verfügung stehenden Tasten und ihre Funktion im Programm. So sieht er immer, welche Taste was bewirkt.

Mit der <Cursor nach oben>-Taste kann er einen Satz zurück blättern, mit <Cursor nach unten> einen Satz nach vorn. <CTRL>-<A> springt direkt zum ersten Satz und <CTRL>-<E> zum letzten. Beachten Sie, daß im Suchmodus nur die Sätze benutzt werden, die auf den Suchbegriff passen. Dann wird immer der Satz angesprungen, der als nächster (letzter) gefunden wurde. <CTRL>-<D> druckt den aktuellen Datensatz auf dem Drucker aus und die <HELP>-Taste ermöglicht es, den dargestellten Satz zu ändern, also zu korrigieren oder durch neue Daten zu ersetzen. <CTRL>-<H> schließlich führt zurück zur Hauptauswahl, sprich zum 'Anfang:'.

In der nachfolgenden Schleife 'SatzLesen:' wird der Dateizugriff erledigt. Da Sie mit den Cursortasten blättern können, muß das Programm zuerst überprüfen, ob die gewünschte 'SatzNummer' zu groß oder zu klein ist. In diesem Fall wird sie auf den gerade noch erlaubten Wert festgelegt. Die Satznummer zeigen wir im rechten oberen Eck des Bildschirms. Der GET-Befehl liest den angegebenen Satz von Diskette in den Datenpuffer.

Jetzt kommen die Routinen zum Suchen von Daten: Ist der Suchmodus aktiv, wird zunächst die Modus-Bezeichnung berichtigt. (Nicht "Daten lesen", sondern "Daten suchen".) Das dann aufgerufene Unterprogramm 'SuchDatenPruefen:' checkt ab, ob der gerade gelesene Satz auf den Suchschlüssel paßt. Falls ja, bekommt 'Gefunden' den Wert 1, falls nein den Wert 0. Nun folgt ein IF/END IF-Block, der sich darum kümmert, was passieren soll, wenn der Satz nicht auf den Suchbegriff paßt: Ist die Obergrenze der Satznummern erreicht, wird 'Richtung' umgekehrt und bekommt den Wert -1. Die Suche findet nun von hinten nach vorn statt. Und zwar solange, bis der letzte passende Satz gefunden wurde.

Dabei kann aber eine Schwierigkeit auftreten: Wenn die aktuelle Datei überhaupt keinen Satz beinhaltet, der auf den Suchschlüssel paßt, würde das Programm die Daten ständig von hinten

nach vorn und wieder von vorn nach hinten durchsuchen. Damit wäre es dann beschäftigt - 24 Stunden, Sonn- und Feiertags an 365 Tagen im Jahr. Kurz: Das Ende der 35-Stunden-Woche für Computer. Aber das kann ja nicht Sinn der Übung sein. Deshalb die nächste Einschränkung: Ist der aktuelle Satz der letzte Satz der Datei, und wurde bisher kein passender Satz gefunden (festzustellen durch 'GefSatz'=0), brechen wir die Suche ab und bringen stattdessen den Hinweis auf den Bildschirm: "Keinen passenden Satz gefunden! Bitte drücken Sie eine Taste." Nach diesem Tastendruck kehren wir zurück zur Hauptauswahl.

Gibt es aber doch mindestens einen passenden Satz, können wir weitermachen: Am unteren Ende der Datei ('SatzNummer'=1) kehren wir die Suchrichtung wieder um ('Richtung'=+1) und suchen nach oben. Die 'Richtung' wird zur 'SatzNummer' addiert und der so berechnete neue Satz gelesen. Hinter dem END IF kann 'GefSatz' auf 1 gesetzt werden, denn nun steht fest, daß mindestens ein passender Satz gefunden wurde.

Danach kommen wieder beide Betriebsarten zu ihrem Recht: werden Zuerst die Feldnamen und dann die Feldinhalte auf den Bildschirm gedruckt. Die aktuellen Satzinhalte merkt sich gleichzeitig wieder das Feld 'Eingabe$', damit diese Daten für Korrekturen zur Verfügung stehen.

Die darauffolgende WHILE...WEND-Schleife wartet auf einen Tastendruck und teilt die gedrückte Taste in 'Tast$' mit. Hat 'Tast$' den Wert CHR$(28), haben Sie also <Cursor nach oben> gedrückt, wird die 'SatzNummer' um eins vermindert und die Suchrichtung wird umgekehrt: Falls der Suchmodus aktiv ist, suchen wir von hinten nach vorn. Bei CHR$(29), dem Code der Taste <Cursor nach unten>, findet dasselbe Spielchen statt, nur mit umgekehrtem Vorzeichen. CHR$(1) wird von <CTRL>-<A> erzeugt. Sie wollen an den Anfang der Datei springen, also erhält 'SatzNummer' den Wert 1.

CHR$(5) entspricht <CTRL>-<E>, also Sprung zum Dateiende. Die Variable 'AnzahlSaetze' beinhaltet die letzte Satznummer. CHR$(8) ist der Code von <CTRL>-<H>. Mit dieser Tastenkombination springen Sie zurück zur Hauptauswahl. Um die nö-

tigen Reiseformalitäten kümmert sich 'EndeLesen:'. Ist doch nicht so schwer, oder? Wenn Sie den aktuellen Satz auf dem Drucker ausdrucken wollen, drücken Sie <CTRL>-<D>. Diese Tastenkombination liefert den Code CHR$(4). Die Feldnamen und Feldinhalte werden mit LPRINT-Befehlen zum Drucker geschickt. Der leere LPRINT hintendran sorgt für den nötigen Abstand zwischen den einzelnen Datensätzen, zumindest auf dem Papier.

Mit der <HELP>-Taste (CHR$(139)) können Sie im aktuellen Satz Korrekturen ausführen oder neue Daten eingeben. "Warum", so denkt sich das Lese-Programm: "soll ich mich damit herumschlagen? Dafür gibt es doch schon einen Programmteil, nämlich 'EingabeKorr:'." Wir haben ja schon vorhin erwähnt, daß sich die Lese-Routine diesen Teil von 'DatenEingeben:' von Zeit zu Zeit ausleiht. Damit 'EingabeKorr:' darüber informiert ist, wer es jetzt eigentlich aufgerufen hat, setzen wir 'DatAend' auf 1. Anhand dieser Variablen weiß das Programm, wohin es nach Beendigung der Korrekturen zurückspringen soll.

Und damit können wir dann die 'Leseschleife:' beenden oder genauer gesagt: sie von Neuem aufrufen. Es schließt sich 'EndeLesen:' an, der Programmteil, der die Heimkehr zur Hauptauswahl regelt: Die Relativ-Datei wird geschlossen, 'Nam$' gelöscht, und schon erfolgt der Sprung zum 'Anfang:'. Jetzt fehlen uns nur noch die Unterprogramme:

```
DatenSuchen:
  CLS : LOCATE 1,1 : COLOR 1,0 : PRINT "Daten suchen"
  LOCATE 1,25 : COLOR 3,0 : PRINT "Datei:";
  COLOR 1,0 : PRINT Nam$
  FOR x=0 TO AnzahlFelder
    LOCATE 5+x,1 : PRINT Feldname$(x)":"
  NEXT x
  COLOR 3,0 : LOCATE 4,1
  PRINT "Bitte geben Sie den Suchwert ein."
  COLOR 1,0
    FOR x=0 TO AnzahlFelder
      LOCATE 5+x,LEN(Feldname$(x))+2
      LINE INPUT Eingabe$
```

```
        IF Eingabe$<>"" THEN
          Such$=LEFT$(Eingabe$,Laenge(x))
          Suchnr=x : x=10
        ELSE
          Such$=""
        END IF
      NEXT x
  Fehlerabfrage3:
      GOSUB FehlerJN
      IF Korr=0 THEN EndeSuchen
      IF Korr=1 THEN SuchKorr
  GOTO Fehlerabfrage3

  SuchKorr:
    LOCATE 5+Suchnr,1 : PRINT Feldname$(Suchnr)":"Such$
    LOCATE 5+Suchnr,LEN(Feldname$(Suchnr))+2
    LINE INPUT Eingabe$
    IF Eingabe$<>"" THEN Such$=LEFT$(Eingabe$,Laenge(Suchnr))
  GOTO Fehlerabfrage3

  EndeSuchen:
    IF Such$="" THEN Suchnr=0 : DatSuch=0
    GefSatz=0
  RETURN

  SuchDatenPruefen:
    x=0
  SuchSchleife:
    x=x+1
    IF x>LEN(Dat$(Suchnr))-LEN(Such$) THEN Gefunden=0 : RETURN
    IF MID$(Dat$(Suchnr),x,LEN(Such$))=Such$ THEN Gefunden=1 : RETURN
  GOTO SuchSchleife

  FehlerJN:
    LOCATE 19,1 : COLOR 3,0
    PRINT "Wollen Sie Fehler korrigieren? (J/N)";
    COLOR 1,0 : INPUT "",a$
    IF UCASE$(a$)="N" OR a$="" THEN Korr=0 : RETURN
    IF UCASE$(a$)="J" THEN Korr=1 : RETURN
  GOTO FehlerJN
```

```
Dat.FelderExistJN:
  OPEN Nam$+".Felder" FOR APPEND AS 1
    IF LOF(1)<=0 THEN DateiExist=0 ELSE DateiExist=1
  CLOSE 1
  IF DateiExist=0 THEN
    LOCATE 3,1 : PRINT SPACE$(60) : BEEP
    LOCATE 3,1 : COLOR 1,0 : PRINT "Datei ";Nam$
    PRINT "existiert nicht!"
    KILL Nam$+".Felder"
    KILL Nam$+".Felder.info"
    Nam$="" : COLOR 3,0
  END IF
RETURN

Dat.FelderLesen:
  FOR x=0 TO 10
    Feldname$(x)="" : Laenge(x)=0
  NEXT x
  OPEN Nam$+".Felder" FOR INPUT AS 2
    INPUT #2,AnzahlFelder
    INPUT #2,Satzlaenge
    INPUT #2,AnzahlSaetze
    FOR x=0 TO AnzahlFelder
      INPUT #2,Feldname$(x)
      INPUT #2,Laenge(x)
    NEXT x
  CLOSE 2
RETURN
```

In 'DatenSuchen:' werden alle Vorbereitungen für den Suchmodus getroffen. In bekannter Weise identifiziert sich der Programmteil und gibt die aktuelle Datei an. Dann erscheinen die Feldnamen auf dem Bildschirm, gefolgt vom Text: "Bitte geben Sie den Suchwert ein.". Sie haben die Möglichkeit, in einem der Felder einen Text einzugeben, auf den das benutzte Datenfeld in jedem Satz überprüft wird.

Wenn Sie also in Ihrer Schallplattensammlung alle Platten von "Elton John" suchen, geben Sie im Feld "Interpret" den Wert "Elton John" ein. Das Programm findet dann alle Sätze, in denen das Feld "Interpret" mit "Elton John" ausgefüllt ist. Würden Sie aber z.B. im Feld "Titel" den Suchtext "Elton John" eingeben, würden bloß die Sätze gefunden, wo auch dieser Text im "Titel" vorkommt. Also z.B. eine Platte namens "Elton John's Best Love Songs". Dabei sehen Sie gleich, daß Sie auch nach Textfragmenten suchen können. Es genügt z.B. im "Interpret"-Feld "John" einzugeben. Allerdings finden Sie dann auch "John Lennon" und "Olivia Newton-John" und auch "Little John und Robin Hood - ein spannendes Abenteuer für Kinder ab 3". Also suchen Sie im Zweifelsfall besser nach "Elton", das kommt wohl kaum in anderen Namen vor.

Groß- und Kleinschreibung werden beim Suchen berücksichtigt. Mit der Eingabe "john" hätten Sie also wahrscheinlich keine Sätze gefunden, höchstens Namen wie "Bigbadjohn". Alles klar? In einer Adreßliste können Sie so auch nach Postleitzahlbereichen oder Telefon-Vorwahlnummern suchen. Wenn Sie nach Leerzeichen <SPACE> oder Einzelbuchstaben, wie "e" suchen, finden Sie ziemlich viele Sätze, aber eben nur die, auf die das Suchmuster paßt.

Und wie haben wir das alles programmiert?

Der 'DatenSuchen:'-Teil läuft durch eine Eingabeschleife. Sie können sich mit <RETURN> solange durch alle Felder des Datensatzes bewegen, bis Sie irgendwo eine Eingabe machen, die kein Leerstring ist. Diese Eingabe wird in der Variablen 'Such$' gespeichert. 'Suchnr' enthält die Nummer des Felds, in dem verglichen werden soll. Die 'Fehlerabfrage3:' bietet, genau wie ihre beiden Kollegen, eine Korrekturmöglichkeit. Allerdings können Sie hier nur noch den Inhalt von 'Such$' verändern, das gewählte Feld ist bereits festgelegt.

'EndeSuchen:' springt zurück zur Zeile, von der das Unterprogramm aufgerufen wurde. Hat 'Such$' keinen Inhalt, wird das Suchen abgeschaltet: 'Suchnr' und 'DatSuch' bekommen den

Wert 0. In jedem Fall auf 0 gesetzt wird die Variable 'GefSatz', damit sich später feststellen läßt, ob nach den zuletzt benutzten Suchkriterien auch wirklich ein Satz gefunden wurde.

'SuchDatenPruefen:' ist der Programmteil, wo der aktuelle Satz dann wirklich auf den Suchbegriff überprüft wird. Die Schleife 'SuchSchleife:' läuft solange, bis alle möglichen Positionen von 'Dat$' mit 'Such$' verglichen sind: Wir bilden aus dem 'Dat$', der zum Feld 'Suchnr' gehört, einen Teilstring mit der Länge des Suchstrings. Dieser Teilstring beginnt im ersten Durchlauf an der ersten Stringposition, im zweiten an der zweiten Position usw. Wird x größer als der verbleibende Rest von 'Dat$', ohne daß eine Übereinstimmung festgestellt wurde, war der 'Such$' in diesem Datensatz nicht zu finden. 'Gefunden' bekommt den Wert 0 und das Unterpogramm springt zurück. Stimmen der mit MID$ gebildete Teilstring und 'Such$' überein, wurde der Suchbegriff gefunden. Als Audruck der allgemeinen Freude im Unterprogramm wird 'Gefunden' auf 1 gesetzt. Für eine Freudenfeier bleibt leider keine Zeit, denn schon folgt der Rücksprung mit RETURN.

Der nächste Programmteil, 'FehlerJN:', wird mehrfach aufgerufen: Hier soll festgestellt werden, ob der Anwender Eingabekorrekturen wünscht. Wir drucken die Frage "Wollen Sie Fehler korrigieren? (J/N)" auf den Bildschirm und warten mit INPUT auf eine Antwort. Durch die etwas ungewohnte Konstruktion INPUT "",a$ unterdrücken wir das Fragezeichen, das sonst bei INPUT automatisch erscheint. Bei der Eingabe von "N" oder einer Leereingabe (durch Drücken von <RETURN>, ohne vorher etwas anderes zu tippen) wird 'Korr'=0: Keine Korrekturen. Den Leerstring akzeptieren wir, um bei der Dateneingabe ein schnelles Arbeiten durch Überwählen sämtlicher Optionen mit der <RETURN>-Taste zu ermöglichen. Haben Sie aber "J" eingegeben, bekommt 'Korr' den Wert 1. Um alles weitere muß sich dann der aufrufende Programmteil kümmern.

Das Unterprogramm 'Dat.FelderExistJN:' haben wir schon mehrfach angekündigt: Anhand der .Felder-Datei überprüft es, ob eine angegebene Datei wirklich existiert. Wenn Sie eine nicht vorhandene Datei zum Lesen öffnen wollen, erhalten Sie einen

"File not found"-Error. Da damit das Programm abgebrochen würde, ist das keine gute Lösung. Was kann man also tun? Wenn Sie eine Datei zum Schreiben öffnen, wird dummerweise der alte Inhalt gelöscht. Auch nicht eben wünschenswert. Was bleibt übrig? Richtig: Der APPEND-Modus. Wenn Sie eine sequentielle Datei FOR APPEND öffnen, werden gegebenenfalls neue Sätze angehängt, aber keine alten gelöscht. Gibt es die Datei noch nicht, wird eine neue Datei erzeugt.

Wir öffnen also die .Felder-Datei FOR APPEND und fragen dann mit LOF(1) die Dateilänge ab: Ist Sie 0 oder -1, stehen noch keine Daten in der Datei. Sie muß also gerade eben erst durch APPEND angelegt worden sein - die Datei existierte vorher noch nicht. Also 'DateiExist'=0. Anderenfalls (ELSE heißt das auf BASIC) 'DateiExist'=1. Hat 'DateiExist' den Wert 0, bringt das Unterprogramm eine Meldung auf den Schirm, daß die gewünschte Datei nicht existiert. Da wir sie durch APPEND neu angelegt haben, wird sie mit KILL gleich wieder gelöscht. Den Inhalt von 'Nam$' können wir getrost vergessen ('Nam$'=""), schließlich gibt es die Datei sowieso nicht. Dann Rücksprung. Um die Bestätigung der Bildschirmausgabe müssen sich die aufrufenden Programme selbst kümmern. Sie sehen, wir neigen in letzter Zeit dazu, zu deligieren - aber das ist ja auch die Aufgabe des Computers, uns Arbeit abzunehmen.

Zu guter Letzt haben wir dann noch das Unterprogramm 'Dat.FelderLesen:'. Wenn es die .Felder-Datei nämlich doch gibt, interessieren wir uns ja für ihren Inhalt. Zuerst löscht eine FOR...NEXT-Schleife die Inhalte von 'Feldname$(x)' und 'Laenge(x)'. Denn falls die gelesene Datei weniger Felder als die zuletzt verwendete hat, würden die alten Feldnamen gespeichert bleiben, was sich später etwas störend auswirkt. Danach werden die Werte für die Anzahl der Felder pro Satz ('AnzahlFelder'), die Gesamtlänge eines Satzes ('Satzlaenge') und die gegenwärtige Anzahl der Sätze in der Datei ('AnzahlSätze') geladen. Außerdem die Feldnamen ('Feldname$(x)') und die Feldlängen ('Laenge(x)'). Anhand dieser Daten kann das Programm dann die Relativ-Datei öffnen und benutzen. Noch ein letzter RETURN-Befehl, dann wäre auch dieses Programm im Kasten.

Falls Sie es selbst eingetippt haben, sei Ihnen hiermit ein großes Lob ausgesprochen. Falls Sie es aber in diesem Fall noch nicht abgespeichert haben, sei Ihnen hiermit ein Tadel ausgesprochen: Sofort nachholen! Und danach ein Dankgebet an den heiligen Stromulus, den Beschützer aller ungespeicherten Daten.

5.3 Wie geht's? - die Bedienungsanleitung zur Datenbank

Alle wichtigen Informationen zur Bedienung des Programms haben Sie eigentlich schon in den Erklärungen erhalten. Wir wollen Ihnen hier nochmal eine zusammenfassende Anleitung bieten, außerdem ein paar hilfreiche Tips zur Arbeit mit Ihrer AmigaBASIC-Datenbank.

Sie ist also für Dateiverwaltung gedacht. Das ist eine der eher geschäftlich-nüchternen Anwendungen für Computer. Aber selbst wenn Sie für's Geschäft keinen Sinn haben, hängt es ja nur von Ihren Daten ab, wieviel Spaß die ganze Sache macht: Sie können Kochrezepte, klassische Lyrik, Videofilme, Bücher, Ihre Zimmerpflanzen, Arbeitskollegen, Fußballvereine, Briefmarkensammlungen, Ihre Amiga-Programme, Hinweise auf Zeitschriftenartikel, moderne Lyrik, Ihre steuerlich absetzbaren Quittungen, Ihre Lieblings-Restaurants, die Schulleistungen Ihrer Sprößlinge und...und...und... in einer Datenbank erfassen. Na, immer noch keine Anregung für Sie dabei? Was die Schulleistungen Ihres Sprößlings betrifft - in diesem Fall sollten Sie dafür Sorge tragen, daß er nie dieses Buch in die Finger bekommt und erfährt, wie man Daten ändert... Sonst könnte sich der Notenschnitt erstaunlich verbessern.

Der erste Schritt auf dem Weg zu einer neuen Datei ist das Einrichten. Dafür gibt es Menüpunkt 1. Sie können bis zu zehn Feldnamen und die gewünschte Länge der Felder eingeben. Höchstlänge sind 40 Zeichen. Dieser Wert wird auch automatisch benutzt, wenn Sie keine Angabe über die Feldlänge machen. Die Eingabe der Feldnamen wird abgebrochen, wenn Sie in der Spalte "Name" keinen Wert eingeben und nur <RETURN> drücken. Bild 16 zeigt Ihnen eine typische Eingabe in diesem Modus:

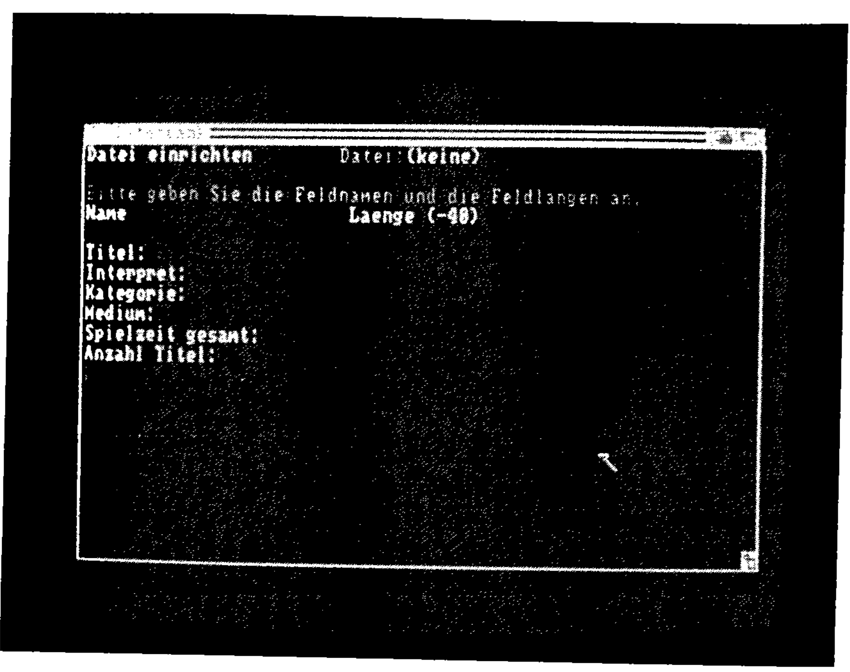


Bild 16: Der Modus "Datei einrichten" im Datenbank-Programm

Nach Beendigung der Eingabe haben Sie die Möglichkeit, eventuelle Tippfehler zu korrigieren: Antworten Sie einfach "J" auf die Frage "Wollen Sie Fehler korrigieren?". Der Cursor erscheint dann wieder im ersten Datenfeld. Geben Sie entweder den neuen Inhalt ein oder drücken Sie <RETURN>. Wenn alles stimmt, tippen Sie den Namen der Datei ein. Das Programm legt dann die beiden nötigen Disketten-Dateien an und kehrt schließlich zur Hauptauswahl zurück.

Wenn Sie nun Option 2 wählen (Neue Daten eingeben), gelangen Sie direkt in den Eingabemodus. Wenn Sie unmittelbar vorher keine neue Datei angelegt haben, müssen Sie erst noch den gewünschten Dateinamen eingeben. Der zuletzt verwendete Name

kann durch "=" oder "*" abgekürzt werden. Das ist besonders praktisch, wenn Sie aus "Daten lesen" oder "Daten suchen" in den Eingabemodus wechseln wollen.

Eine sog. Maske (sie wird aus Ihren Feldnamen aufgebaut) erscheint auf dem Bildschirm. Der Eingabecursor steht im ersten Datenfeld des aktuellen Satzes. Geben Sie die Daten ein, Tippfehler können danach wieder korrigiert werden. Ein völlig leerer Satz wird nicht abgespeichert, sondern nochmal zur Eingabe angeboten. Wenn Sie auf die Frage "Weitere Sätze?" mit "J" oder einer leeren Eingabe antworten, gelangen Sie zur nächsten Eingabe. Bei "N" wird die Datei geschlossen und Sie kehren zurück ins Hauptprogramm.

Option 3 dient zum Anzeigen der Inhalte einer Datei. Geben Sie den Dateinamen ein, eventuell durch "=" oder "*" vertreten. Sollte es die Datei nicht geben oder sie keine Sätze enthalten, weist Sie das Programm darauf hin. Anderenfalls erscheint der erste Datensatz auf dem Schirm. Unterhalb des Datenbereichs sehen Sie in oranger Schrift die möglichen Tasten und ihre Funktion. Sie können vorwärts und rückwärts in den Sätzen blättern, direkt zum ersten oder letzten Satz springen, den Satzinhalt auf dem Drucker ausgeben oder auf dem Bildschirm verändern. Mit <CTRL>-<H> springen Sie zurück zur Hauptauswahl. Bild 17 zeigt als Beispiel einen Satz aus unserer Schallplattendatei:

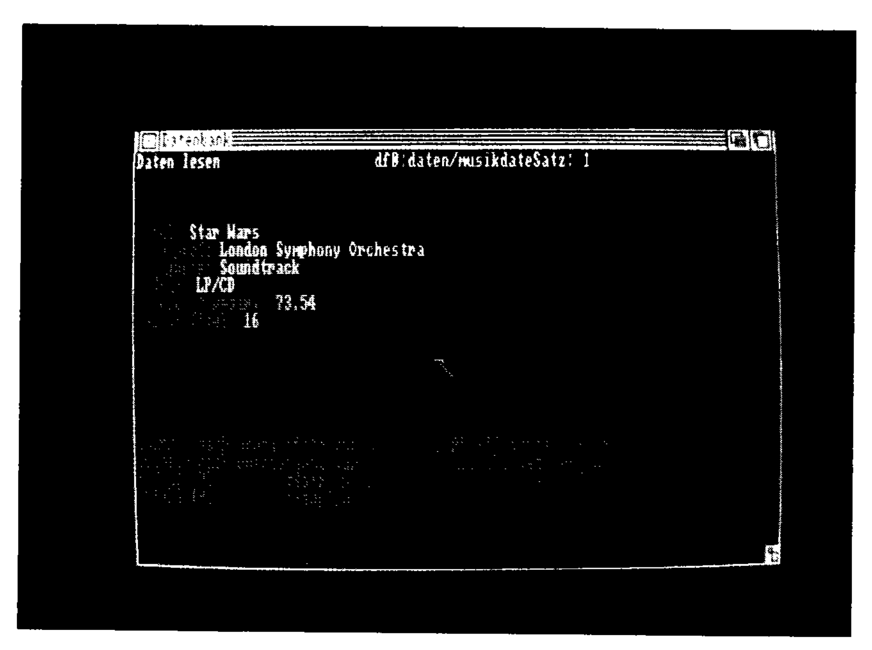


Bild 17: Der Modus "Daten lesen" im Datenbank-Programm

Mit Option 4, "Daten suchen", können Sie alle Sätze nach einem bestimmten Suchkriterium durchsuchen lassen. Nur die Sätze, auf die das Kriterium zutrifft, werden angezeigt. Nach der Wahl von "Daten suchen" geben Sie bitte zuerst den Namen der Datei an, die durchsucht werden soll. Das Programm druckt daraufhin die Feldnamen auf den Bildschirm und bittet Sie, in eines der Felder einen Suchwert einzugeben. Auf diesen Suchwert werden alle Datensätze geprüft. Nach der möglichen Korrektur von Tippfehlern öffnet das Programm die Datei und beginnt mit der Suche. Wenn mindestens ein Satz die Suchbedingung erfüllt, zeigt das Programm die gefundenen Sätze an. Anderenfalls erscheint der Text "Keinen passenden Satz gefunden!". Mit den gefunden Sätzen können Sie arbeiten wie im Lese-Modus auch. Nur werden in diesem Modus ausschließlich die gefundenen Sätze angezeigt, alle anderen überspringt das Programm.

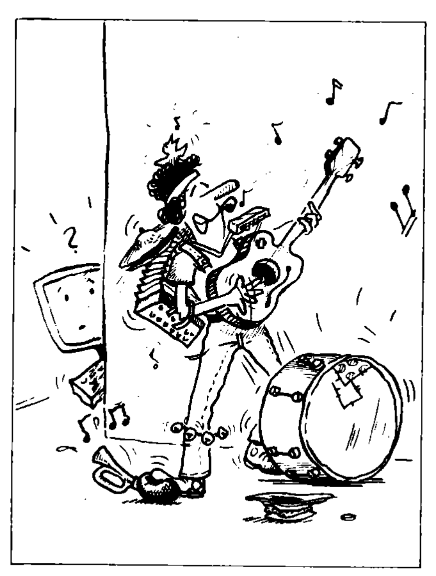
Das war's. Wir wünschen noch viel Spaß auf der Jagd nach speicherbaren Daten. Vielleicht verstehen Sie unseren Innenminister jetzt ja sogar ein wenig besser: Datenbanken sind doch wirklich lustig, oder? Aber an dieser Stelle ist vielleicht wirklich doch einmal eines der wenigen ernsten Worte in diesem Buch angebracht: Es heißt nämlich nicht umsonst "Datenbank". Mit Ihren persönlichen Daten sollten Sie genauso vorsichtig umgehen wie mit Ihrem Geld. Denn es kann schnell dazu kommen, daß irgendjemand mit Ihren Daten Mißbrauch treibt. Und manche persönliche Dinge gehen außer Sie wirklich niemanden etwas an.

Computer können nichts für die Daten, die man ihnen eingibt: Ein Computer ist letzten Endes bloß ein Werkzeug. Ein sehr leistungsfähiges und faszinierendes Werkzeug zwar, aber eben nur ein Werkzeug. Die Menschen allein tragen die Verantwortung und seiner Verantwortung muß auch hier jeder gerecht werden.

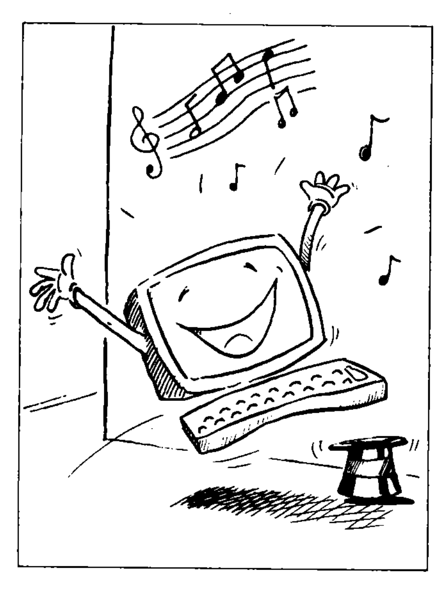
Sogar für Ihr kleines AmigaBASIC-Datenbank-Programm sollten Sie eine Art von Daten-Moral entwickeln: Das Einkommen Ihres Nachbarn oder der Name der Freundin seines Sohnes hat auf Ihren Disketten nichts verloren. Und was Du nicht willst, daß man Dir tu...

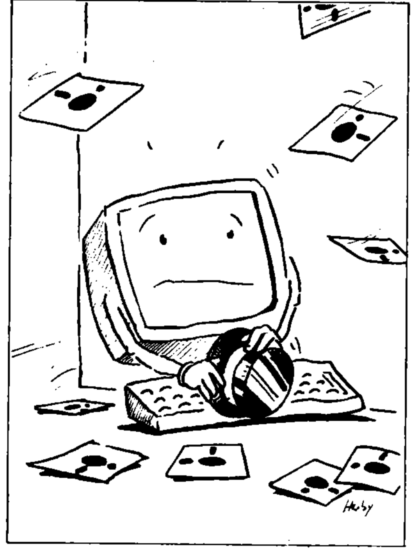
Mit diesen Worten als Ausklang nähern wir uns dem dritten großen Teil unseres Buches. Das heißt aber nicht, daß wir fortan mit Daten nichts mehr zu tun haben werden. Ganz im Gegenteil. Sie werden sich gleich wundern, mit wievielen Daten aller Art der Weg zu Computersprache und Computermusik gepflastert ist. Wir bitten um Ruhe im Konzertsaal, die Ouvertüre beginnt.

III. Der Sound-Amiga









6. Amiga calling - Spracherzeugung in BASIC

Ihr Amiga macht ja im allgemeinen nicht gerade einen schüchternen Eindruck. Trotzdem war die Kommunikation mit ihm bisher ein bißchen einseitig. Das soll sich bald ändern. Denn die Tatsache, daß Amiga nur spricht, wenn er darum gebeten wird, heißt nicht, daß er nichts zu sagen hätte. Der Amiga hat alle nötigen Hardware- und Softwarevoraussetzungen, um erstaunlich verständliche Computersprache zu erzeugen. Vielleicht haben Sie ihn ja schon mal reden hören. Bei den Workbench-Demos gibt es entsprechene Programme und auch in der "BasicDemos"-Schublade auf der "Extras"-Diskette finden sich Beispiele für seine Beredtheit.

Voraussetzung für den ganzen folgenden Teil unseres Buchs ist natürlich, daß Sie den Amiga auf irgendeine Weise an einen Lautsprecher angeschlossen haben. Der eingebaute Lautsprecher des Monitors ist völlig ausreichend. Natürlich ist eine 200W-Stereoanlage auch nicht von Nachteil... Wo auch immer, bitte stellen Sie den Lautstärkeregler auf eine Ihnen angenehme Lautstärke. (Vielleicht denken Sie dabei auch ein bißchen an Ihre Nachbarn. Wir werden nämlich später auch Geräusche ausprobieren. Und ein versehentlich ausgelöster Bombenalarm regt nur die wenigsten Menschen zum Lachen an...) Sollten Sie diesen Teil des Buchs gegen Mitternacht lesen, bitte wir Sie ganz besonders, auf Ihre schlafenden Nachbarn und die dann herumschwirrenden armen Geister und Vampire Rücksicht zu nehmen.

6.1 Die dritte Kostprobe: Amiga spricht mit tausend Zungen

Hier ist mal wieder eine unserer Kostproben. Vielleicht haben Sie unsere kleinen Appetithäppchen in den Daten-Kapiteln vermißt. Aber zum Thema "Dateiverwaltung" war es fast unmöglich, mit wenigen Befehlen eine eindrucksvolle Demonstration vom Amigas Fähigkeiten vorzuführen. Dafür übergeben wir das Wort jetzt auch gleich unserem Amiga:

```
Text$="Amiga"
Anfang:
  Feld%(0)=90+180*RND
  Feld%(1)=RND
  Feld%(2)=50+200*RND
  Feld%(3)=RND
  Feld%(4)=17000+10000*RND
  Feld%(5)=40+24*RND
  Feld%(6)=11*RND
  Feld%(7)=0
  Feld%(8)=0

  SAY TRANSLATE$(Text$),Feld%
GOTO Anfang
```

Dieses Programm ist als "Kostprobe3" in der "Sprache"-Schublade unserer Diskette im Buch zu finden.

Wenn Sie das Programm selbst eingegeben haben und es abspeichern wollen, könnten Sie damit gleich Ihre Schublade "Sprache" auf Ihrer Version der "BasicDisk" einweihen. (Bitte aber keine Champagnerflaschen gegen den Amiga werfen. Schicken Sie die lieber an uns...) Der erste Befehl vor dem Abspeichern ist also: CHDIR ":Sprache"

Nach Starten des Programms braucht AmigaBASIC erst mal die Workbench. Wenn Sie nur ein einziges Diskettenlaufwerk besitzen oder die Workbench in keinem Laufwerk eingelegt ist, wird ein Dialog-Window um die Workbench bitten. Nach kurzem Laden geht's dann los. Ihr Amiga sagt in allen möglichen Stimmlagen, Sprechgeschwindigkeiten, Tonhöhen und Lautstärken seinen Namen.

Er ist schon ziemlich vielseitig, unser Computer, nicht wahr? Wenn Sie nicht wollen, daß er immer nur von sich erzählt, können Sie ja mal andere Werte für 'Text$' eingeben. Aber wundern Sie sich nicht, wenn's meistens sehr amerikanisch klingt.

Wollen Sie lernen, wie auch Ihr Amiga zu einer Art Gisela Schlüter unter den Computern wird? Dann lesen Sie bitte das nächste Kapitel.

6.2 Gespräche auf Bitebene - die Befehle SAY und TRANSLATE$

In der Kostprobe haben Sie schon die beiden wichtigsten Befehle zur Spracherzeugung in Aktion gesehen: SAY und TRANSLATE$. Der SAY-Befehl (deutsch: "Sag, sprich") dient zur Ausgabe von künstlicher Sprache. Wenn Sie aber mal im BASIC-Window versuchen, Ihren Amiga mit

```
say "Amiga"
```

zum Sprechen zu überreden, erleben Sie zunächst einigen Widerstand: "Illegal function call". Ganz so einfach geht's nämlich auch wieder nicht. Der SAY-Befehl kann nicht direkt die Inhalte von Strings vorsprechen.

Zuerst müssen die Texte in eine ihm verständliche Form übersetzt werden. "Übersetzen" heißt auf Englisch "to translate". Daher hat die Funktion TRANSLATE$ Ihren Namen:

```
Text$=TRANSLATE$("Amiga")
SAY Text$
```

TRANSLATE$ übersetzt den Text in eine für SAY sprechbare Form (einen sogenannten Phonem-Code, aber dazu kommen wir etwas später). Muten wir dem Amiga doch mal eine längere Ansprache zu. Am besten begrenzen Sie mit

```
width 80
```

erstmal den Darstellungsbereich des Bildschirms. Sonst müssen Sie einen Teil des folgenden Satzes nämlich blind eingeben, weil er hinter dem rechten Bildschirmrand verschwindet. Ist diese Vorbereitung erledigt, gehen wir frisch ans Werk:

```
Text$="This is your Amiga Computer speaking. I feel good, how are you?"
Text$=translate$(Text$)
say Text$
```

Man muß sich schon ein wenig konzentrieren, wenn man die Sprachausgabe verstehen will, ohne den Text zu kennen. Trotzdem ist sie nach einer kleinen Gewöhnungsphase ziemlich gut verständlich. Wenn Sie den Text gerade eben nicht verstanden haben, kann das natürlich auch daran liegen, daß Sie kein Englisch können. Damit wären wir dann bei einem weiteren Problem. Als gebürtiger Amerikaner spricht der Amiga nun mal nicht besonders gut Deutsch. Probieren Sie's ruhig mal aus:

```
Text$="Hallo, hier spricht Ihr Amiga. Wie geht es Ihnen?"
Text$=translate$(Text$)
say Text$
```

Die erste Hälfte des Satzes klingt ja noch ganz gut, aber "Wai dschätt äs innen?" ?!? Das erinnert zwar enorm an einen Römer nach der liebevollen Behandlung durch Asterix und Obelix, aber nicht an korrektes Deutsch.

Wie vorhin schon erwähnt, versteht der SAY-Befehl nur Texte in einer bestimmten Codierung. Diese Codierung (Verschlüsselung) nennt man Phonem-Code. Wenn Sie mal sehen wollen, wie der SAY-Befehl Ihr "Wie geht es Ihnen?" empfängt, dann geben Sie mal ein:

```
Text$="Wie geht es Ihnen?"
Text$=translate$(Text$)
say Text$
? Text$
```

Auf dem Bildschirm erscheint dann:

```
WAY4 JEH4T EH4Z IH4NEHN.
```

So sieht ein Phonem-String aus. Er ist das Ergebnis der Übersetzung von "Wie geht es Ihnen?" durch die Funktion TRANSLATE$. Das ist auch nicht weiter verwunderlich, denn so etwa

hört sich unser Satz nach englischen bzw. amerikanischen Betonungs- und Ausspracheregeln an. Und TRANSLATE$ ist nur dafür gedacht, englischen Text in Phonem-Code zu übersetzen. Der Schuldige ist also gar nicht der SAY-Befehl, sondern die Übersetzungsfunktion.

Wenn man mit AmigaBASIC deutsche Sprache erzeugen will, gibt es dafür drei mögliche Ansätze: Der erste wäre, die TRANSLATE$-Funktion so umzuschreiben, daß sie nach deutschen Ausspracheregeln übersetzt. Ob Commodore und Microsoft gewillt sind, diese Arbeit zu erledigen, steht zur Zeit noch nicht verbindlich fest. Wünschenswert wäre es auf jeden Fall. Der zweite Weg ist eine Abwandlung des ersten: Er besteht nämlich darin, den deutschen Phonem-Code selbst zu erzeugen. Dazu haben wir später noch ein paar Anmerkungen. Der dritte Weg ist, die amerikanische Übersetzungsroutine einfach so auszutricksen, daß das Ergebnis mehr oder weniger deutsch klingt. Ein Beispiel:

```
Text$=translate$("Vee kaiht ass eenan?")
say Text$
```

Um das einigermaßen professionell hinzubekommen, muß man aber doch recht gut über englische Aussprache Bescheid wissen. (Na, war das nicht wieder ein perfekt eingebautes Eigenlob?) Oder man muß beim Ausprobieren verschiedener Versionen sehr viel Geduld aufbringen. "Vee kaiht ass eenan?" soll "Wie geht es Ihnen?" heißen. Mal ehrlich: Wären Sie darauf gekommen? Schauen Sie sich noch mal die Phonem-Code-Übersetzung an:

```
? Text$
```

liefert

```
VIY4 KEY3T AE4S IY4NAEN.
```

"Phonem" ist ein Begriff aus der Sprachlehre und bedeutet "Laut". Der Phonem-Code ist also eine Lautschrift. Die hat mit der korrekten Schreibweise eines Worts wenig zu tun, sie richtet sich vielmehr danach, wie die Laute gesprochen werden. V steht

für ein stimmhaftes "W" wie in "Wasser" oder "Vase". Hier sehen Sie gleich, daß verschiedene Buchstaben durchaus den gleichen Laut vertreten können. IY ist das Phonem für ein langes "i", wie in "wie" oder "Biene". Und die Zahl 4 dahinter ist ein Betonungswert. Er kann zwischen 0 und 9 liegen und gibt an, wie stark der Laut betont wird.

Versuchen wir uns doch einmal an einem eigenen Phonem-String:

```
Text$="VIY9 GEY3T AE1S IY6NAEN?"
say Text$
```

Das "wie" haben wir sehr stark betont, das "es" etwas abgeschwächt und dafür wieder mehr Nachdruck auf "Ihnen" gelegt. Wie finden Sie das Ergebnis?

Die Schreibweise in Phonem-Code folgt festen Regeln, man kann nicht einfach nach Gefühl die Laute durch Buchstaben ersetzen. Wenn Sie Lust haben, sich näher damit zu beschäftigen, empfehlen wir Ihnen den Anhang H aus dem Amiga-BASIC-Handbuch von Commodore. Dort wird sehr ausführlich erklärt, wie der Phonem-Code aufgebaut ist.

Die Firma Commodore hat auch ein BASIC-Programm geschrieben, das versucht, deutschen Text in Phonem-Code zu übersetzen. Es befindet sich unter dem Namen "speechd" in der "Basic-Demos"-Schublade der "ExtrasD"-Diskette. Damit können Sie auch immer mal wieder spicken, wenn Sie eigene Texte auf Deutsch sprechen lassen wollen. Näheres zu diesem Programm erfahren Sie im Anhang C, wo wir die Beispielprogramme aus der "BASICDemos"-Schublade der "Extras"-Diskette erklären.

Wir wollen uns zunächst mal auf englische Texte beschränken. Und hoffen, daß es irgendwann doch einmal eine deutsche TRANSLATE$-Routine gibt. Dazu müßte möglicherweise gar nicht mal viel an AmigaBASIC selbst verändert werden. Denn die Sprach- und Übersetzungsroutinen liegen unter dem Namen "narrator.device" ("Erzähler") auf der Workbench. Diese Routinen sind es, die bei der ersten Anwendung von SAY und TRANS-

LATE$ geladen werden. Wenn Sie ein Sprachprogramm neu starten, verlangt Ihr Amiga auch sofort wieder die Workbench. Der Grund dafür liegt darin, daß AmigaBASIC die Übersetzungsroutinen so bald wie möglich wieder los werden möchte. Immerhin besetzen sie fast 30 KByte im Arbeitsspeicher. Jedes NEW und jeder RUN-Befehl bewirkt das Löschen der "narrator.device"-Routinen aus dem Speicher. Bei Bedarf müssen sie halt wieder neu geladen werden.

Manchmal kommt es auch vor, daß die Übersetzungsroutinen gelöscht wurden, aber AmigaBASIC das nicht so richtig mitbekommen hat. Es geht dann davon aus, es habe sie noch im Speicher. Die Zusammenarbeit zwischen AmigaBASIC und den Workbench-Routinen funktioniert leider nicht immer ganz reibungslos. Dann verursacht jeder TRANSLATE$-Befehl und jeder SAY-Befehl einen "Illegal function call"-Error. Wir haben gegen dieses Problem leider auch kein anderes Rezept, als die Workbench neu zu laden. Vergessen Sie nicht, vorher alles abzuspeichern.

Das Problem tritt nach unserer Erfahrung dann am häufigsten auf, wenn Sie nach dem Löschen eines Sprachprogramms die Sprachbefehle im Direktmodus verwenden. Um Ihre Sprach-Programme müssen Sie sich also keine Sorgen machen.

Und jetzt lassen wir mal wieder den Amiga zu Wort kommen.

6.3 Die Sprechstunde - Optionen beim SAY-Befehl

Das eigentlich Interessante an der Kostprobe aus Kapitel 6.1 war die Vielfalt der Stimmen, mit denen AmigaBASIC sprechen kann. Hoch oder tief, langsam oder schnell, betont oder monoton, männlich oder weiblich. Wenn Sie ein bißchen Geduld hatten, haben Sie eindrucksvolle und teilweise sehr witzige Beispiele gehört. Fragt sich, wie man diese Stimm-Steuerung selbst programmieren kann. Schwer ist es nicht, wie wir Ihnen gleich zeigen werden. Nur die Vielzahl der Optionen und die Breite der möglichen Regelbereiche wirkt am Anfang etwas verwirrend.

Bitte löschen Sie das Programm im LIST-Window, denn wir wollen Stück für Stück ein kleines Testprogramm schreiben.

Zunächst das Grundprinzip: Hinter dem Text können Sie bei SAY ein Integer-Feld mit mindestens 9 Elementen angeben. Hier mehr als 9 Elemente anzugeben, würde übrigens nichts nützen, da der überzählige Speicherplatz zwar verloren geht, aber AmigaBASIC die angegebenen Werte einfach ignorieren wird. Er sieht sich immer nur die ersten 9 Zahlen an. Geben Sie also zunächst im LIST-Window ein:

```
DIM Feld%(8)
```

Sie wissen ja noch: DIM Feld%(8) ermöglicht 9 Elemente, weil Element Nummer 0 mitzählt. Wenn Sie den DIM-Befehl in eigenen Sprachprogrammen weglassen, ist das auch nicht tragisch. Durch die automatische Dimensionierung auf 10 Bytes gehen 2 Bytes verloren. Was soll's, wir sind ja großzügig.

In jedem der 9 Feldelemente steht eine Zahl, die für eine bestimmte Eigenschaft oder einen Modus der erzeugten Stimme steht. Wir wollen diese Zahlen aus DATA-Zeilen einlesen:

```
FOR x=0 TO 8
  READ Feld%(x)
NEXT x
```

Nun brauchen wir einen Text, der gesprochen werden soll:

```
Spracheingabe:
  LINE INPUT "Text:",Eingabe$
  IF Eingabe$<>"=" THEN Text$=Eingabe$
  Sprech$=TRANSLATE$(Text$)
  CLS : PRINT Text$
  SAY Sprech$,Feld%
GOTO Spracheingabe
```

Wir haben das Programm so konstruiert, daß Sie durch die Eingabe von "=" den letzten Satz wiederholen können. Und jetzt kommt's. In der folgenden DATA-Zeile verleihen wir dem Amiga sozusagen ein neues akustisches Image:

```
DATA 140,0,160,0,22000,64,11,1,0
```

Probieren Sie das Programm ruhig gleich aus. Sie werden hören, daß sich die Stimme von Amiga doch reichlich verändert hat. Sie klingt etwas höher, weicher als die Original-Stimme. Die Änderung hängt natürlich mit den DATA-Werten zusammen. Also schauen wir uns mal an, was es da so gibt.

Der erste Wert im Integerfeld gibt die Grundfrequenz der Stimme an. Dieser Wert entscheidet hauptsächlich über die Stimmhöhe und damit über den wichtigsten Wert der Stimmeigenschaften. Erlaubt sind für die Grundfrequenz Werte zwischen 65 (sehr, sehr tief) und 320 (eine richtige Piepsstimme). Der Standardwert, den AmigaBASIC benutzt, ist 110. Wir haben uns für 140 entschieden, liegen also ein Stück höher. Für technisch Interessierte: Hier wird einfach die Grundfrequenz in Hertz angegeben.

Verändern Sie doch mal das Listing und geben Sie hier zunächst den Wert 80 ein. Klingt recht sonor, finden Sie nicht? Probieren Sie auch mal die Extremwerte 65 (der klingt schon fast ein bißchen schaurig) und 320 (das ist so hoch, daß es unnatürlich klingt.) Setzen Sie danach die Zahl für unsere Experimente mit den anderen Werten wieder auf eine Frequenz, die Ihnen gefällt.

Der nächste Wert kann nur 0 oder 1 sein. Hier wird die Betonung geregelt. 0 bedeutet normale, modulierte Sprechweise und 1 erzeugt eine völlig betonungsfreie Computerstimme. Natürlich klingt die menschliche Betonung schöner, aber wenn Sie mal ausdrücklich einen Roboter sprechen lassen oder Ihrer Familie die Steigerung vom monotonen Computer zum Amiga vorführen wollen: Jetzt wissen Sie, wie's geht. Die AmigaBASIC-Grundeinstellung für diesen Wert ist 0.

Der dritte Wert motiviert nun wirklich zum Spielen: Mit ihm steuern Sie die Sprechgeschwindigkeit. Die Zahl gibt Wörter pro Minute an. Von AmigaBASIC voreingestellt ist 150, wir lassen in unseren DATA-Zeilen etwas schneller reden: 160. Probieren Sie mal den Satz "Fishers Fritz fisht frisha Fisha" (Liebe Deutschlehrer! Bitte verzeihen Sie uns diesen Anschlag auf die Rechtschreibung...) mit der Einstellung 350, und dann machen Sie's nach.

Gültige Werte für die Sprechgeschwindigkeit liegen zwischen 40 und 400. Bei sehr hohen Geschwindigkeiten besteht allerdings die Gefahr, daß der Amiga sich beim Sprechen selbst überholt und dann Silben verschluckt. Und sehr niedrige Werte (so unter 70) sind ein hervorragendes Schlafmittel: Bis ein Satz gesprochen ist, hat jede mittelmäßig begabte Rennschnecke die Strecke Flensburg-München zurückgelegt - ähem, na zumindest auf einer Landkarte.

Kommen wir zum vierten Wert. Sie können hier wählen, ob der Amiga mit einer männlichen oder einer weiblichen Stimme spricht. (Ach übrigens: Kennen Sie den sprechenden Bordcomputer aus "Raumschiff Enterprise", der sich in Captain Kirk verliebt hat? Scheint ein entfernter Verwandter von Amiga zu sein.) 0 erzeugt eine männliche Stimme und 1 eine weibliche. Damit das Ganze wirklich einigermaßen weiblich klingt, müssen Sie auch die Grundfrequenz anpassen. Versuchen Sie mal eine Grundfrequenz von 240 Hertz.

Und auch der nächste Wert sollte für die weibliche Stimme modifiziert werden. Setzen Sie ihn mal auf 23000. Dieser fünfte Wert legt die Abtastfrequenz oder ("Samplingfrequenz") fest. Da Sie sich darunter wahrscheinlich noch nichts vorstellen können, wollen wir uns zunächst merken, daß er im Zusammenspiel mit der Grundfrequenz die Stimmhöhe beeinflußt. Der Wert darf im Bereich von 5000 bis 28000 liegen. AmigaBASICs Grundwert ist 22200, und unserer liegt nur etwas tiefer. Versuchen Sie hier zunächst nur vorsichtige Veränderungen, schon die haben eine große Wirkung.

Beim nächsten, sechsten Wert wird's schon wieder einfacher: Hier geben wir eine Lautstärke zwischen 0 und 64 ein. Wie AmigaBASIC benutzen auch wir die höchste Lautstärke: 64. Bei der Programmierung von Unterhaltungen ist dieser Wert sehr nützlich. Bedenken Sie aber, daß der Lautstärkeregler an Ihrem Monitor oder Ihrer Stereoanlage am längeren Hebel sitzt. Er steuert nämlich die Gesamtlautstärke.

Der siebte Wert legt die Zuordnung der Sprachausgabe zu einem oder mehreren Tonkanälen fest. Das merken Sie nur, wenn Sie Ihren Amiga irgendwie in Stereo wiedergeben können. Der Monitorlautsprecher ermöglicht nämlich nur Monowiedergabe und schickt alles auf einen Lautsprecher. Der Amiga ist aber voll stereotauglich. Es gibt insgesamt vier Tonkanäle, die fest dem linken oder dem rechten Kanal zugeordnet sind. Die Kanäle mit den Nummern 0 und 3 (also der höchste und der niedrigste Kanal) sind mit dem linken der beiden Tonausgänge am Amiga verbunden, die beiden mittleren Kanäle 1 und 2 mit dem rechten Ausgang. Bei bewußter Steuerung der Kanalzuordnung können Sie Stereoeffekte erzielen, zumindest aber ein Zwiegespräch von links und rechts.

Zum Thema "Stereo" gibt es einige Einschränkungen. Der Raumklang-Effekt bei einer Stereo-Aufnahme basiert ja nicht darauf, daß aus beiden Kanälen exakt dasselbe Signal kommt (wie beim Amiga), sondern ein leicht verändertes Signal. Wenn Sie den Amiga über Kopfhörer genießen, wirkt die strenge Kanaltrennung unnatürlich. Kommt ein Ton von links, ist der rechte Kanal völlig tot. Auch das ist bei Stereoaufnahmen anders. Da hört man rechts wenigstens einen abgeschwächten Anteil des linken Kanals. Läuft der Amiga über Stereolautsprecher, sieht's schon besser aus, weil sich dann der Schall im Raum ausbreiten kann und akustischen Gesetzen wie Reflexionen etc. unterliegt.

Es gibt für Sprache elf mögliche Kanalzuordnungen, Sie sehen Sie in Tabelle 10.

6. Wert im Integer-Feld	Kanal/Kanäle	Anmerkungen
0	Kanal 0	links
1	Kanal 1	rechts
2	Kanal 2	rechts
3	Kanal 3	links
4	Kanäle 0 und 1	Stereo
5	Kanäle 0 und 2	Stereo
6	Kanäle 3 und 1	Stereo
7	Kanäle 3 und 2	Stereo
8	alle freien linken Kanäle	0 und/oder 3
9	alle freien rechten Kanäle	1 und/oder 2
10	jedes freie Paar	AmigaBASICs Grundwert
11	jeder freie Kanal	

Tabelle 10: Kanalzuordnungen beim SAY-Befehl

Zu den "freien Kanälen": Hier geht's nicht um die Bundespost und ihr Kabelprojekt, sondern um die Tatsache, daß auch die Tonerzeugung beim Amiga dem Multitasking unterliegt. Von anderen Programmen (oder sogar von vorherigen BASIC-Befehlen, dazu gleich mehr) kann bereits ein Teil der Kanäle belegt sein. Dabei geht es aber nur um tatsächliche, gerade gespielte Töne. Ein Kanal gilt als frei, wenn auf ihm kein Ton läuft. Je nach dem angegebenen Wert können Sie Ihre Sprache auf beliebige Kanalkombinationen schicken. Wenn noch andere SAY-Befehle oder Ton-Befehle laufen (Sie werden noch einige kennenlernen...), sollten Sie auf jeden Fall eine der Optionen wählen, die "freie" Kanäle benutzt. So kommt die Sprache am sichersten auch wirklich beim Zuhörer an.

AmigaBASIC benutzt als Standardwert 10 (jedes unbesetzte Paar links/rechts). Wir haben in unserer DATA-Zeile den Wert 11 gewählt (jeder freie Kanal). Aber wie gesagt: Experimente lohnen sich nur bei Stereo-Anschluß.

Die letzten beiden Werte im Integer-Feld geben an, wie sich der Amiga verhalten soll, wenn mehrere SAY-Befehle aufeinander folgen. Oder SAY-Befehle und andere Befehle. Ist der achte Wert 0, wartet AmigaBASIC mit der Ausführung weiterer Befehle, bis die Ausgabe des SAY-Befehls beendet ist. 1 bewirkt, ähnlich wie bei den OBJECT-Befehlen, ein Anstarten der Tonausgabe und macht dann mit weiteren Befehlen weiter. So kann Sprache im Hintergrund laufen, während AmigaBASIC anderen Aufgaben nachgeht. AmigaBASIC versucht sich aber solch stressige Doppelbelastungen möglichst vom Hals zu halten, deshalb ist die Grundeinstellung 0.

Wert 9 spielt nur dann eine Rolle, wenn der achte Wert auf 1 steht, also auf Hintergrundverarbeitung. Wie soll sich AmigaBASIC verhalten, wenn während einer laufenden Sprachausgabe ein neuer SAY auftritt? 0 bewirkt eine Art Warteschlange: Der Neue soll gefälligt warten, bis der alte fertig ist. 2 bewirkt, daß die laufende Ausgabe abgebrochen wird, und der neue SAY-Befehl sofort beginnt. 1 haben wir mit voller Absicht hintenan gestellt: Dieser Wert hat mit den Optionen 0 und 2 nichts zu tun: Er deaktiviert den aktuellen SAY-Befehl, und zwar völlig unabhängig von allen anderen Einstellungen. Das können Sie im Rahmen größerer Programme dazu benutzen, eine SAY-Ausgabe durch eine Variable (deren Inhalt 0 für Reden und 1 für Schweigen ist - Sie kennen ja die Geschichte mit Silber und Gold) ein- und auszuschalten.

So, das wär's erstmal. Es waren ganz schön viele Optionen, nicht wahr? Erfahrungsgemäß geht jetzt die große Suche nach den schönsten Einstellungen los. Um Ihnen dabei viel Tipparbeit und lange Wartezeiten zu ersparen (bei jedem Programmstart wird ja erst mal von neuem "narrator.device" geladen), stellen wir Ihnen im nächsten Kapitel ein Hilfsprogramm zur Stimmen-Programmierung vor.

6.4 Gesagt, getan - das Sprachutility

Unser nächstes Utility hilft Ihnen beim Ausprobieren der verschiedenen Stimm-Optionen. Wenn Sie die Stimme gefunden haben, die Sie sich schon immer wünschten, können Sie die Daten abspeichern und später via MERGE an eigene Programme anhängen. Ein Hinweis noch: Die in diesem Programm produzierten Stimmen haben weder bei Landtags- noch bei Bundestagswahlen Gültigkeit und dürfen auch nicht abgegeben werden. Und alle, die lieber klicken statt tippen, können auch das Programm "Sprachutility" von der beiliegenden Diskette laden. Aber bitte schauen Sie sich trotzdem die Erklärungen an. Wie immer.

```
Bildschirmaufbau:
  CLS
  PALETTE 0,.1,.1,.4
  LOCATE 2,2 : PRINT "Text:"
  LINE (60,7)-(612,18),1,b
  LOCATE 22,6
  PRINT "Freq.    Geschw.    Abtast    Lautst."
  LINE (40,30)-(65,160),1,b
  LINE (120,30)-(145,160),1,b
  LINE (205,30)-(230,160),1,b
  LINE (285,30)-(310,160),1,b
  LOCATE 6,54 : PRINT "Männl.        Weibl."
  LINE (420,30)-(495,48),1,b
  LINE (510,30)-(585,48),1,b
  LOCATE 9,54 : PRINT "Mensch        Comp."
  LINE (420,57)-(495,75),1,b
  LINE (510,57)-(585,75),1,b
  LOCATE 12,59 : PRINT "Sprechen"
  LINE (450,84)-(555,102),1,b
  LOCATE 17,59 : PRINT "Speichern"
  LINE (450,120)-(555,138),1,b
```

Der Programmteil 'Bildschirmaufbau:' baut ein System aus Reglern und Feldern auf. Sie kennen ja schon unseren Spleen mit Bildschirmfarben: Auch diesmal wird das Workbench-Blau vornehm abgedunkelt.

Im obersten Feld können Sie den Text eingeben, der gesprochen werden soll. Darunter stehen vier Schieberegler für die Grundfrequenz, die Sprechgeschwindigkeit, die Abtastfrequenz und die Lautstärke. All diese Werte können innerhalb ihres zulässigen Bereichs völlig verändert werden. Neben den Reglern gibt es noch ein Felderpaar für männliche oder weibliche Stimmausprägung und ein Paar für die Festlegung von betonter (menschlicher) oder unbetonter (computer-typischer) Textwiedergabe. Ein Feld aktiviert das Sprechen mit den gegenwärtig eingestellten Werten. Und das letzte Feld können Sie anklicken, wenn Sie die DATA-Werte abspeichern wollen. Bild 18 zeigt den vollständigen Bildschirmaufbau, so können Sie feststellen, ob in Ihrer Version alles stimmt.

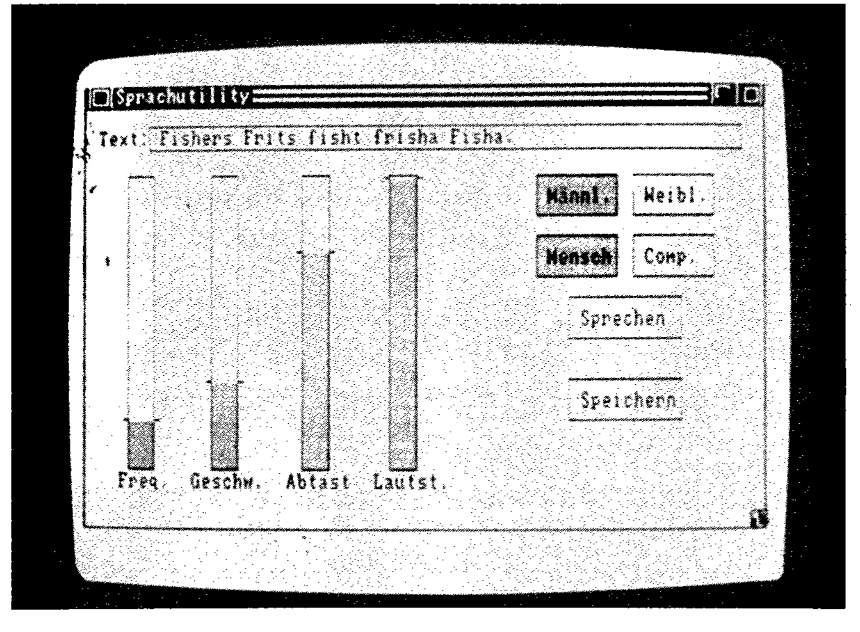


Bild 18: **Das Sprachutility**

```
Ausgangswerte:
  FOR x=0 TO 8
    READ Sprech%(x)
  NEXT x
  DATA 110,0,150,0,22200,64,10,0,0
  GOSUB WerteZeigen

Hauptschleife:
  ON MOUSE GOSUB Mausauswertung
  MOUSE ON
  WHILE 1 : WEND
```

In 'Ausgangswerte:' lesen wir ins Integerfeld 'Sprech%' die AmigaBASIC-Grundwerte für den SAY-Befehl ein. Im Lauf des Programms können Sie alle diese Werte ändern, aber zu Beginn haben sie diese Ausgangsbelegung. Wenn Sie irgendwann einmal besonders liebgewonnene Einstellungsdaten für SAY haben, können Sie diese gern in der entprechenden DATA-Zeile angeben. Das Programm arbeitet mit jeder beliebigen Grundeinstellung. Wichtig ist nur, daß überhaupt eine existiert. Im Anschluß daran rufen wir das Unterprogamm 'WerteZeigen:' auf, das die Regler und Markierungen richtig setzt. Die 'Hauptschleife:' zeigt Ihnen, daß wir wieder mal ein Programm geschrieben haben, das vollständig über Event Trapping läuft. Allerdings benutzen wir nur die Maus - Pulldowns werden vom Programm nicht abgefragt.

```
Mausauswertung:
  Test=MOUSE(0)
  x=MOUSE(3) : y=MOUSE(4)
  IF x>39 AND x<311 AND y>29 AND y<161 THEN
    IF x<66 THEN
      Frequenz:
      Sprech%(0)=(255-(y-30)*(255/130))+65
      FreqWert=((320-Sprech%(0))/255)*130
      LINE (41,31)-(64,31+FreqWert),0,bf
      LINE (41,32+FreqWert)-(64,159),3,bf
```

```
  y=MOUSE(6)
  IF y<31 THEN y=31
  IF y>159 THEN y=159
  IF MOUSE(0)<=-1 THEN Frequenz
END IF
```

Den letzten Teil (ab IF x<66...) sollten Sie - wenn Sie das Programm selbst eingeben - mit "Copy" aus dem "Edit"-Menü ausschneiden und dreimal mit "Paste" kopieren. Er wird mit kleineren Korrekturen insgesamt viermal verwendet. Die nächsten drei Blocks folgen jetzt, dann geht das Programm normal weiter.

```
IF x>119 AND x<146 THEN
  Geschw:
  Sprech%(2)=(360-(y-30)*(360/130))+40
  GeschwWert=((400-Sprech%(2))/360)*130
  LINE (121,31)-(144,31+GeschwWert),0,bf
  LINE (121,32+GeschwWert)-(144,159),3,bf
  y=MOUSE(6)
  IF y<31 THEN y=31
  IF y>159 THEN y=159
  IF MOUSE(0)<=-1 THEN Geschw
END IF
IF x>204 AND x<231 THEN
  Abtast:
  Sprech%(4)=(23000-(y-30)*(23000/130))+5000
  AbtastWert=((28000-Sprech%(4))/23000)*130
  LINE (206,31)-(229,31+AbtastWert),0,bf
  LINE (206,32+AbtastWert)-(229,159),3,bf
  y=MOUSE(6)
  IF y<31 THEN y=31
  IF y>159 THEN y=159
  IF MOUSE(0)<=-1 THEN Abtast
END IF
IF x>284 AND x<311 THEN
  Lautst:
  Sprech%(5)=(64-(y-30)*(64/130))
  LautstWert=((64-Sprech%(5))/64)*130
  LINE (286,31)-(309,31+LautstWert),0,bf
  LINE (286,32+LautstWert)-(309,159),3,bf
```

```
      y=MOUSE(6)
      IF y<31 THEN y=31
      IF y>159 THEN y=159
      IF MOUSE(0)<=-1 THEN Lautst
    END IF
  END IF
  IF x>419 AND x<496 AND y>29 AND y<49 THEN
    Sprech%(3)=0
    PAINT (422,32),3,1 : PAINT (512,32),0,1
  END IF
  IF x>509 AND x<586 AND y>29 AND y<49 THEN
    Sprech%(3)=1
    PAINT (422,32),0,1 : PAINT (512,32),3,1
  END IF
  IF x>419 AND x<496 AND y>56 AND y<76 THEN
    Sprech%(1)=0
    PAINT (422,59),3,1 : PAINT (512,59),0,1
  END IF
  IF x>509 AND x<586 AND y>56 AND y<76 THEN
    Sprech%(1)=1
    PAINT (422,59),0,1 : PAINT (512,59),3,1
  END IF
  IF x>59 AND x<613 AND y>6 AND y<19 THEN
    LOCATE 2,9 : PRINT SPACE$(54)
    LOCATE 2,9 : LINE INPUT Text$
  END IF
  IF x>449 AND x<556 AND y>83 AND y<103 THEN
    PAINT (452,85),3,1
    SAY TRANSLATE$(Text$),Sprech%
    PAINT (452,85),0,1
  END IF
  IF x>449 AND x<556 AND y>119 AND y<139 THEN
    PAINT (452,121),3,1
    LOCATE 2,9 : PRINT SPACE$(54)
    LOCATE 2,9 : COLOR 0,3 : PRINT "Dateiname:";
    COLOR 1,0 : LINE INPUT Nam$
    IF Nam$<>"" THEN
      IF Nam$= "=" OR Nam$="*" AND Altname$<>"" THEN Nam$=Altname$
      OPEN Nam$ FOR OUTPUT AS 1
        PRINT #1, "REM DATAs erzeugt mit AmigaBASIC-Sprachutility"
```

```
        PRINT #1, "DATA ";
        FOR x=0 TO 7
          PRINT #1,Sprech%(x)",";
        NEXT x
        PRINT #1,Sprech%(8)
      CLOSE 1
      Altname$=Nam$
    END IF
    LOCATE 2,9 : PRINT SPACE$(54)
    LOCATE 2,9 : COLOR 1,0 : PRINT Text$
    PAINT (452,121),0,1
  END IF

RETURN
```

In der 'Mausauswertung:' arbeitet das Programm den größten Teil der Zeit. Zu Beginn rufen wir MOUSE(0) auf und weisen das Ergebnis einer Dummy-Variablen namens 'Test' zu: Wir wollen nur möglichst aktuelle Werte über die Mausposition. Um die x- und die y-Koordinaten der Maus zu erfahren, benutzen wir die Funktionen MOUSE(3) und MOUSE(4). Wenn Sie sich nicht mehr erinnern, schauen Sie ruhig im Befehlsanhang oder auch im Kapitel 2.9 nach. MOUSE(3) und MOUSE(4) liefern die Koordinaten beim Drücken der Maustaste, also beim Unterprogrammaufruf. Wir benutzen diese Werte, weil wir uns bei einer Mausbewegung nicht für die aktuelle Position interessieren, sondern für die Position beim Klick.

Nun werden der Reihe nach die verschiedenen Felder auf dem Bildschirm abgeprüft. Wo fand der Klick statt? War es irgendwo im Reglerbereich? Dann gibt es für jeden der Regler einen Programmteil ('Frequenz:', Geschw:', 'Abtast:' und 'Lautst:'). Die vier sind völlig gleich aufgebaut, unterscheiden sich aber in den Zahlen, Daten und Namen.

Wir wollen die Funktion anhand des ersten Blocks ('Frequenz:') einmal durchexerzieren, die restlichen drei schenken wir uns und Ihnen, da sie völlig identisch funktionieren. Liegt die Mausposition also im Bereich des Frequenz-Reglers, errechnen

wir zuerst aus der y-Koordinate der Maus eine neue Frequenz und schreiben sie ins Feld 'Sprech%(0)'. Die Variable 'FreqWert' errechnet die Bildschirmkoordinate, die dieser Frequenz im Regler entspricht. Der erste der beiden LINE-Befehle mit Blockfill-Option füllt den Bereich oberhalb der Oberkante des Reglers in seiner aktuellen Stellung blau aus. Der zweite LINE-Befehl füllt den Bereich unterhalb der Reglerkante orange aus. So entsteht ein Schieberegler, der, wie die Quecksilbersäule in einem Thermometer, mit wachsenden Werten nach oben steigt. 'y' erhält die aktuelle y-Position der Mausbewegung. Liegt 'y' außerhalb des Reglerbereichs (31 bis 159), wird es auf den Minimal- bzw Maximalwert gesetzt. Ist MOUSE(0) eine Zahl kleiner oder gleich -1, ist die Maustaste noch gedrückt. In diesem Fall wird der Regler weiterbewegt, also Rücksprung zum Label 'Frequenz:'.

Wie gesagt, wiederholt sich diese Prozedur mit veränderten Daten für die anderen Regler. Danach folgt die Abprüfung des Kästchens "Männlich". Haben Sie dort hinein geklickt, bekommt 'Sprech%(3)' den entsprechenden Wert, nämlich 0. Das "Männlich"-Feld wird ausgemalt, das "Weiblich"-Feld in normale Farbe zurückversetzt. Der nächste Block prüft, ob es nicht vielleicht doch das "Weiblich"-Kästchen war, wo der Klick stattfand. Sollte das zutreffen, bekommt 'Sprech%(3)' den Wert 1, das "Männlich"-Kästchen wird zurückgesetzt und das "Weiblich"-Kästchen ausgemalt.

Der dritte Block in dieser Art ist für das "Mensch"-Kästchen zuständig, also für die Auswahl der betonten Sprechweise. 'Sprech%(1)' wird mit 0 versehen, die Kästchen werden wieder nach dem aktuellen Stand ausgemalt: "Mensch" an, "Computer" aus. Natürlich nur im Sinne von Ausmalen. Sie brauchen Ihren Amiga nicht auszuschalten. Und der vierte in diesem Bunde ist das "Computer"-Kästchen, das genau das Gegenteil des "Mensch"-Kästchens bewirkt: 'Sprech%(1)' wird 1, und das "Computer"-Kästchen bekommt die orange Füllung.

Der nun folgende Teil ermöglicht die Eingabe des Strings, der gesprochen werden soll. Beachten Sie bitte, daß diese Eingabe mit <RETURN> beendet werden muß, bevor Event Trapping normal weiterarbeiten kann. Liegen die Koordinaten des Klick-Punkts im "Sprechen"-Kästchen, erzeugt ein SAY-Befehl aus allen vorliegenden Daten den gewünschten (oder unerwünschten?!) Ohrenschmaus.

Der nächste Block ist wieder etwas länger. Er beinhaltet die Speicherung der DATA-Datei. Zur Eingabe des Dateinamens benutzen wir einfach das Text-Fester: Wir schaffen Platz mit SPACE$ und lesen mit LINE INPUT den Namen ein ("=" und "*" dienen wieder als Abkürzung). War der Name nicht leer, öffnen wir die Datei.

Was wir hier machen, ist schon recht trickreich: Eine sequentielle Datei und ein Programm im ASCII-Format unterscheiden sich nicht voneinander. Wir schreiben also eine Reihe von Befehlen in die Datei und können diese Befehle später mit MERGE an ein eigenes Programm anhängen.

Ehre, wem Ehre gebührt. Deshalb die REM-Zeile, in der wir darauf hinweisen, wie und wodurch die DATAs entstanden sind. In die nächste Zeile der Datei kommt der DATA-Befehl und dann acht durch Kommas getrennte Einzelwerte aus dem Feld 'Sprech%(x)'. Den neunten Wert schicken wir solo hinterher, da er nicht mehr von einem Komma gefolgt werden darf. Das war auch schon alles, die Datei kann wieder geschlossen werden.

'Altname$' merkt sich noch den letzten Dateinamen für die Abkürzung bei nächsten Mal. Jetzt müssen wir uns nur noch für die Gastfreundschaft im Text-Fenster revanchieren und alles so hinterlassen, wie wir es vorgefunden haben. Ein SPACE$ löscht den Dateinamen und der alte 'Text$' wird wieder an seine Stelle gedruckt. PAINT setzt das "Schreiben"-Kästchen zurück auf blaue Farbe. Während des Schreibvorgangs war es die ganze Zeit orange gefüllt.

Das RETURN am Schluß springt aus der 'Mausauswerten:'-Routine zurück in die 'Hauptschleife:'. Dort macht sich ja die endlose WHILE...WEND-Schleife ein schönes Leben. Nichts zu tun, diese Programme heutzutage...

Uns fehlt noch das Unterprogramm 'WertZeigen:', das die Reglerpositionen und Feldauswahlen beim Programmbeginn auf den Bildschirm bringt:

```
WerteZeigen:
  LOCATE 2,9 : PRINT SPACE$(54)
  LOCATE 2,9 : PRINT Text$
  IF Sprech%(3)=0 THEN
    PAINT (422,32),3,1 : PAINT (512,32),0,1
  ELSE
    PAINT (422,32),0,1 : PAINT (512,32),3,1
  END IF
  IF Sprech%(1)=0 THEN
    PAINT (422,59),3,1 : PAINT (512,59),0,1
  ELSE
    PAINT (422,59),0,1 : PAINT (512,59),3,1
  END IF

  FreqWert=((320-Sprech%(0))/255)*130
  LINE (35,31+FreqWert)-(70,31+FreqWert)
  LINE (41,31)-(64,31+FreqWert),0,bf
  LINE (41,32+FreqWert)-(64,159),3,bf

  GeschwWert=((400-Sprech%(2))/360)*130
  LINE (115,31+GeschwWert)-(150,31+GeschwWert)
  LINE (121,31)-(144,31+GeschwWert),0,bf
  LINE (121,32+GeschwWert)-(144,159),3,bf

  AbtastWert=((28000-Sprech%(4))/23000)*130
  LINE (200,31+AbtastWert)-(235,31+AbtastWert)
  LINE (206,31)-(229,31+AbtastWert),0,bf
  LINE (206,32+AbtastWert)-(229,159),3,bf
```

```
LautstWert=((64-Sprech%(5))/64)*130
LINE (280,31+LautstWert)-(315,31+LautstWert)
LINE (286,31)-(309,31+LautstWert),0,bf
LINE (286,32+LautstWert)-(309,159),3,bf

RETURN
```

Wenn Ihnen die Routinen aus diesem Unterprogramm bekannt vorkommen, ist das kein Wunder: Sie stehen in derselben oder ähnlicher Form auch im Programmteil 'Mausauswertung:'. Es ließ sich aber nicht vermeiden, diese Routinen nochmal in einem eigenen Unterprogramm zusammenzufassen, damit der Bildschirmaufbau beim Programmstart vollständig ist. Würden wir auf das Unterprogramm 'WerteZeigen:' verzichten, würden die einzelnen Regler erst in den Augenblick auf dem Bildschirm erscheinen, wenn sie zum ersten Mal benutzt werden. Bei jedem Regler bringen wir zu Beginn des Programms an der Stelle, an der die Standardeinstellung liegt, eine Markierung an. Das erleichtert später das Wiederfinden dieser Grundeinstellungswerte.

Falls Sie in der DATA-Zeile im Programmteil 'Ausgangswerte:' eigene Lieblingswerte eingeben, zeigen die Markierungen diese Werte an. Was immer dort in den DATAs steht, gilt für das Programm als Standardwert.

Vergessen Sie nicht, das Programm abzuspeichern, wenn Sie es selbst eingegeben haben. Die Bedienung des Utilities ist ganz einfach: Benutzen Sie die Maus, um die Regler zu verschieben. Nach einem Klick ins Textfeld können Sie dort den Text eingeben, der gesprochen werden soll. Aber bitte erst <RETURN> drücken, bevor Sie weitere Mausaktionen starten. Event Trapping merkt sich zwar die Mausklicks, kann sie aber nicht bearbeiten, solange ein INPUT aktiv ist.

Sie können in den dafür vorgesehenen Feldern wählen, ob die Stimme männlich oder weiblich klingen soll, außerdem die Betonung festlegen. Ein Klick ins Feld "Sprechen" liest Ihnen den

Text im Textfeld mit den aktuellen Sprach-Parametern vor. Vor dem ersten Vorlesen ist wieder mal ein Workbench-Zugriff fällig.

Wenn Ihnen die eingestellte Stimme gefällt, können Sie die aktuellen Werte durch einen Klick ins "Speichern"-Feld abspeichern. Es wird ein Programm in ASCII-Format erzeugt, das Sie mit MERGE an ein eigenes Programm anhängen können.

Speichern Sie jetzt bitte das Sprachutility mit seinen letzen Korrekturen ab. Jetzt noch zur Demonstration: Stellen Sie eine hübsche Stimme ein und legen Sie die Daten auf Diskette ab. Dateiname könnte z.B. "Sprachdaten" sein. Löschen Sie dann das LIST-Window und geben Sie ein kleines Beispielprogramm ein:

```
FOR x=0 TO 8
  READ a%(x)
NEXT x
SAY "Hello Amiga.",a%
```

Nun folgt im BASIC-Window der MERGE-Befehl:

```
merge "Sprachdaten"
```

Daraufhin werden zwei neue Zeilen an Ihr Programm angehängt:

```
REM DATAs erzeugt mit AmigaBASIC-Sprachutility
DATA .....
```

In der DATA-Zeile stehen alle Daten über die Stimme, die Sie mit dem Utility ausgesucht haben.

Zum Schluß noch ein kleiner Tip: Wenn Sie das Utility während der Entwicklung eines größeren eigenen Programms benötigen, benutzen Sie doch einfach die RAM-Disk! Geben Sie als Dateinamen "RAM:Sprachdaten" ein und MERGEn Sie später aus der RAM-Disk. Das geht sehr schnell und belegt nicht Ihre Diskette. Wollen Sie die kleinen DATA-Programme aufheben, müssen Sie sie natürlich aus der RAM-Disk zurück auf Diskette kopieren. Sie könnten sich eine kleine Stimmensammlung auf Diskette

einrichten! Dann können Sie nach einiger Zeit aus einem reichen Fundus schöpfen, wenn eines Ihrer Programme mal was zu sagen hat.

Damit sind wir auch schon am Ende der Spracherzeugung. Jetzt geht es noch um Musik und Töne. Unsere Ohren sind ja durch die langen Unterhaltungen mit Amiga schon geschult und verwöhnt.

7. Tatatataaa - Töne und Musik

In unserem Kapitel über die Sound-Möglichkeiten des Amiga darf natürlich neben der Sprache auch die Musik nicht fehlen. Die Hardware, die den Amiga zum Reden bringen kann, ist so leistungsfähig, daß auch sehr komplexe Töne für sie keine Schwierigkeit darstellen. Allerdings müssen wir Sie auch gleich auf ein Problem hinweisen. Die Leistungsfähigkeit der Hardware wird in diesem Fall nicht hundertprozentig von AmigaBASIC unterstützt. Während bisher eigentlich fast alle Möglichkeiten des Amiga durch komplexe und mächtige BASIC-Befehle sehr weit ausgeschöpft werden konnten, beschränkt sich die Ausgabe von Musik und Geräuschen auf zwei eher schmalbrüstige Vertreter ihrer Zunft. Aber dazu später mehr.

7.1 Die vierte Kostprobe: Star Wars - die Musik

In den nächsten Kapiteln wird sich alles um die Erzeugung von Musik in AmigaBASIC drehen. Natürlich kann man hier nicht die Ergebnisse erreichen, die Sie vielleicht von professionellen Musikprogrammen oder den Workbench-Demos kennen, aber das nächste Programm zeigt Ihnen, was in AmigaBASIC ohne großen Aufwand möglich ist:

```
Lesen:
  READ Freq,Dauer
  IF Freq=-1 THEN SOUND RESUME : END
  SOUND WAIT
  SOUND Freq,Dauer,21,0
  SOUND Freq/2,Dauer,127,1
  SOUND Freq*2,Dauer,21,2
GOTO Lesen

DATA 523.25,15,784,15,698.48,6
DATA 659.28,6,587.28,6,1046.52,15
DATA 784,17,-1,-1
```

Wir haben uns erlaubt, einen unserer Lieblingsfilme zu bemühen, genauer gesagt, die Musik daraus. Sie hören die ersten Takte der Filmmusik zu Star Wars. Wenn Sie die Zeilen selbst eingetippt haben und die Musik nicht so harmonisch klingt, wie Sie es von Ihrem letzten Kinobesuch in Erinnerung haben, prüfen Sie doch bitte nochmal die DATA-Zeilen. Vielleicht hat sich irgendwo ein Tippfehler eingeschlichen.

7.2 Hier spielt die Musik - der SOUND-Befehl

Beim Spielen von Noten oder Tönen aller Art spielt der SOUND-Befehl die Hauptrolle. Ihn haben Sie schon an verschiedenen Stellen in unseren bisherigen Programmen kennengelernt. Im Malprogramm zum Beispiel, wo SOUND zum Erzeugen eines Warntons benutzt wird. So eine Aufgabe ist ja fast ein bißchen zu anspruchslos für den Künstler SOUND. Aber wie jeder freischaffende Künstler nimmt er halt jeden Job an, der ihm angeboten wird.

Für Warntöne gibt es ja auch den BEEP-Befehl. Er erzeugt den bekannten Piepser, der dem Anwender anzeigt, daß irgendetwas Besonderes los ist. BEEP erzeugt einen Ton mit einer festen Frequenz (Tonhöhe) und einer festen Dauer. Im Gegensatz dazu können Sie beim SOUND-Befehl die Frequenz und die Dauer beliebig wählen. SOUND kann zwar noch einiges mehr, aber dazu kommen wir dann etwas später.

Wie Sie sicher schon wissen, sind Töne und Geräusche nichts anderes als Luftschwingungen, die vom menschlichen Ohr wahrgenommen und verarbeitet werden. Die Frequenz eines Tons ist die Anzahl der Schwingungen pro Sekunde. Viele Schwingungen bewirken einen hohen Ton, wenige Schwingungen einen niedrigen Ton. Die Einheit für "Schwingungen pro Sekunde" nennt man Hertz, nach dem Physiker, der die elektromagnetischen Wellen entdeckte. Wenn Sie sich darunter nicht viel vorstellen können: Ihm haben wir unter anderem Radio und Fernsehen zu verdanken. Dank seiner Entdeckung können Signale auf vielen verschiedenen Frequenzen rund um die Welt geschickt werden. Oder auch nur vom Funkturm bis zu Ihrer Antenne. Ein Radio,

ein Kassettenrecorder, einfach jede Komponente einer Stereoanlage setzt die zu erzeugende Musik in elektrische Signale um, die zu einem oder mehreren Lautsprechern geschickt werden. Im Lautsprecher versetzt ein Elektomagnet eine Membran in Schwingung, diese Schwingung wird an die Luft weitergegeben - wir hören einen Ton.

Genauso funktioniert die Tonerzeugung beim Amiga. Der Chip, der für Töne und Geräusche zuständig ist (er heißt Paula, auch über ihn - oder sie? - hören wir bald noch Näheres), erzeugt ein entsprechendes Signal an einem oder beiden Tonausgängen. Dieses Signal muß wie bei einer Stereoanlage noch verstärkt werden, dann kann es zum Lautsprecher wandern. Wenn Sie nur den Amiga-Monitor zur Tonwiedergabe verwenden: Auch in ihm sind ein kleiner Verstärker und ein kleiner Lautsprecher.

Nach so viel Theorie wieder etwas mehr Praxis: Bei SOUND können Sie die Frequenz eines Tons direkt angeben. Der Amiga kann Töne zwischen 20 und 15000 Hertz erzeugen. Mit diesem Frequenzumfang kann er sich neben jeder mittleren bis guten Stereoanlage sehen lassen. Eine zweiter Wert ist noch nötig, nämlich die Dauer des Tons. Hier sind Werte zwischen 0 und 77 erlaubt. 0 erzeugt gar keinen Ton, 1 einen extrem kurzen und 77 einen ca. 4 Sekunden langen Ton. Man kann die Dauer genauer berechnen, aber das ist für uns im Moment erst mal unwichtig.

```
sound 440,18
```

erzeugt einen 440-Hertz-Ton, der ca. 1 Sekunde gespielt wird. Für Musiker und solche, die es werden wollen: 440 Hertz ist die Frequenz des "Kammertons a". Warum dieser Ton ausgerechnet und ausschließlich in Kammern gespielt werden soll, konnte bisher noch nicht eindeutig geklärt werden.

```
sound 500,10
```

spielt einen etwas höheren, etwas kürzeren Ton. Wenn Sie wissen wollen, welche Werte hinter unserem bekannten BEEP stecken, probieren Sie doch mal

```
sound 880,3
```

880 Hertz entsprechen übrigens dem Ton a aus der nächsthöheren Oktave. Wollen Sie mal den gesamten Frequenzumfang Ihres Amiga hören? Kein Problem:

```
for x=20 to 15000 step 10 : sound x,1 : next x
```

Tun Sie sich und Ihren Ohren einen Gefallen, und drehen Sie den Lautstärkeregler etwas nach unten. Ein interessantes akustisches Phänomen ist, daß das menschliche Ohr die mittleren Frequenzen lauter hört als extrem hohe und extrem niedrige Frequenzen. Und sehr hohe Frequenzen, entsprechend laut gespielt, können sehr unangenehm wirken. Den STEP 10 haben wir eingebaut, weil wir sonst mehrere Minuten auf den Amiga warten müßten. Sie können diesen Befehl bei Interesse ja auch mal rausnehmen.

Mit dem bisherigen Wissen können Sie schon Piepser und Töne aller Art erzeugen. Um den richtigen Ton zu treffen, hilft bisher nur Ausprobieren. Aber das wird sich bald ändern.

Vielleicht finden Sie es etwas unbequem, ständig Ihren Lautstärkeregler verändern zu müssen. Am Monitor geht's ja noch, aber Ihre Stereoanlage steht möglicherweise am anderen Ende des Zimmers. Und jedesmal dorthin zu laufen... Auch für dieses Problem bietet AmigaBASIC eine Lösung: Den dritten Wert beim SOUND-Befehl. Hier können Sie eine Lautstärke angeben, ähnlich wie bei SAY. Nur liegt diesmal der erlaubte Bereich zwischen 0 (kein Ton) und 255 (volle Lautstärke). Wenn Sie keinen Wert für die Lautstärke angeben, nimmt AmigaBASIC den Grundwert 127 an. Also die goldene Mitte. Vergleichen Sie

```
sound 880,10,48
```

und

```
sound 880,10,255
```

Ja, unser Amiga kann ein ganz schöner Schreihals sein. Hören wir uns mal denselben Ton mit zunehmender Lautstärke an:

```
for x=0 to 255 : sound 880,1,x : next x
```

Auch diese Lautstärkeregelung ist schon bei Warntönen recht hilfreich. So können Sie einen Ton mehr oder weniger aufdringlich machen. Für Musik ist er geradezu unverzichtbar, schließlich ist die Dynamik (das Verhältnis von lauten und leisen Passagen) ein wichtiger Aspekt bei der Musikwiedergabe.

Einen vierten, letzten Wert können Sie bei SOUND auch noch angeben: Die Kanalzuordnung. Auf welchem der vier Tonkanäle soll der Ton gespielt werden? Im Gegensatz zum SAY-Befehl können Sie aber immer nur einen Kanal (0, 1, 2 oder 3) angeben. Sie erinnern sich sicher: 0 und 3 waren links, 1 und 2 rechts. AmigaBASIC benutzt als Grundeinstellung den Kanal Nummer 0.

Die Zuordnung der Seiten können Sie nur verfolgen, wenn Ihr Amiga seine Töne in Stereo wiedergibt. Aber die vier Kanäle nutzen Ihnen auch in Mono: So können Sie bis zu vier verschiedene Töne gleichzeitig spielen. In unserer Kostprobe benutzen wir zum Beispiel drei Töne. Um zu gewährleisten, daß die Töne auch gleichzeitig erklingen, müssen Sie noch zwei weitere Befehle kennenlernen. Gedulden Sie sich bitte noch ein wenig.

Wenn Sie die Werte für Lautstärke und Kanalzuordnung miteinander verbinden, lassen sich interessante Effekte erzielen. Ein Ton, der auf einem Kanal immer leiser wird und auf dem anderen immer lauter, bewegt sich scheinbar von links nach rechts bzw. von rechts nach links. Probieren Sie's mal im LIST-Window:

```
FOR x=0 TO 255
  SOUND 440,1,x,0
  SOUND 440,1,255-x,1
NEXT x
```

So könnten Sie ja mal die akustische Version des letzten Spiels Boris Becker gegen Ivan Lendl programmieren: Bumm links, Bumm rechts, Bumm links...

Jetzt kennen wir uns zwar schon ganz gut aus mit SOUND, aber sehr melodisch waren unsere Beispiele bisher noch nicht. Wenn Sie Noten spielen wollen, brauchen Sie Informationen über die Frequenzen. Unser Kammerton a hat die Frequenz 440 Hertz. Schauen wir uns mal die Oktave an, in der dieser Ton liegt. ("Oktave" nennt der Musiker den Tonbereich von einem c zum nächsthöheren c. Zumindest vom hohen C hat ja jeder schon mal in der einen oder anderen Weise gehört.) Tabelle 11 gibt darüber Auskunft.

Ton-bezeichnung	Frequenz in Hertz
c	261.63
d	293.66
e	329.63
f	349.23
g	392.00
a	440.00
h	493.88
c	523.25

Tabelle 11: **Frequenzwerte einer Oktave**

Sollten sich in der geschätzten Leserschaft Musikexperten befinden, die Ihren Amiga jetzt gern zum Stimmen nutzen würden, müssen wir eine Kleinigkeit erklären: AmigaBASIC erzeugt aufgrund der Hertzangabe eine Schwingung. Dazu werden verschiedene interne Schwingungen des Amiga benutzt. In der gegenwärtigen Version von AmigaBASIC richtet sich die Berechnung der Frequenzen nach den Schwingungen in der amerikanischer Amiga-Version. Auf dem deutschen Amiga klingen die Töne deshalb minimal verstimmt. Wer kein geschultes Gehör besitzt, wird den Unterschied aber wohl kaum bemerken.

Eigentlich haben Sie jetzt schon alles, was Sie zum Musikmachen brauchen. Die Berechnung von Frequenzen läuft nämlich nach einem einfachen Prinzip: Die Frequenz eines Tons verdoppelt sich mit jeder Oktave. Das a in der nächsthöheren Oktave hat also 880 Hertz, das f 698.46 Hertz und so weiter. Wenn Sie die Werte halbieren, erhalten Sie die Frequenzen der nächsttieferen Oktave. Das a hat dort zum Beispiel 220 Hertz. Tiefere Frequenzen als die 130.82 Hertz des c aus dieser tiefsten Oktave können Sie zum Musikmachen kaum verwenden, da Sie unter 100 Hertz keine Töne, sondern nur noch Brummen oder Knacken hören. Um Ihnen die ganze Umrechnerei zu ersparen, gibt's in Tabelle 12 eine Übersicht über die gebräuchlichsten Notenwerte.

Tonbezeichnung, Oktave Nummer	Frequenz in Hertz	Tonbezeichnung, Oktave Nummer	Frequenz in Hertz
c,1	130.82	c,3	523.28
d,1	146.83	d,3	587.28
e,1	164.82	e,3	659.28
f,1	174.62	f,3	698.48
g,1	196.00	g,3	784.00
a,1	220.00	a,3	880.00
h,1	246.94	h,3	987.76
c,2	261.63	c,4	1046.52
d,2	293.66	d,4	1174.52
e,2	329.63	e,4	1318.52
f,2	349.23	f,4	1396.92
g,2	392.00	g,4	1568.00
a,2	440.00	a,4	1760.00
h,2	493.88	h,4	1975.52
c,3	523.25	c,5	2093.00

Tabelle 12: Noten und Frequenzen für den SOUND-Befehl

Das c mit 2093.00 Hertz ist schon ziemlich hoch. Höhere Frequenzen werden Sie für Ihre Musik auch kaum benutzen.

Jetzt könnten Sie die Noten aus der DATA-Zeile unserer Kostprobe herausfinden, indem Sie die angegebenen Frequenzen in unserer Tabelle suchen. Beim "Komponieren" von Musik geht's genau umgekehrt: Sie kennen die Note und suchen die Frequenz.

Fehlen Ihnen nur noch Angaben über die Tondauer. Ein Musiker gibt die Länge eines Tons nicht in Sekunden, sondern in Zeittakten an. Und solchen Zeittakten entspricht die Längenangabe im SOUND-Befehl. Ziemlich genau 18 Zeittakte entsprechen einer Sekunde. In Tabelle 13 die ganz exakten Werte:

Länge in Sekunden	Wert im SOUND-Befehl
0.1	1.8
0.2	3.6
0.4	7.3
0.5	9.1
0.6	10.9
0.8	14.6
1.0	18.2
2.0	36.4
3.0	54.6
4.0	72.8

Tabelle 13: Tonlängen in Sekunden und die zugehörigen Angaben bei SOUND

Anhand dieser Tabelle können Sie jede Tonlänge errechnen. Wenn Sie 3.5 Sekunden brauchen, addieren Sie einfach den Wert für 3.0 und den Wert für 0.5 Sekunden: 54.6 + 9.1 = 63.7. Eine Längenangabe von 63.7 erzeugt also einen Ton von dreieinhalb Sekunden Dauer. In der Kostprobe haben wir die Tonlängen nach Gefühl angegeben. Experimentieren Sie ruhig ein wenig mit den Tempi in unserem Programm.

Und wenn Sie Lust haben, können Sie sich ja jetzt mal an die Komposition von "Alle meine Entchen" oder (für Fortgeschrittene) Beethovens Fünfter machen.

7.3 As Time goes by - SOUND WAIT und SOUND RESUME

Der SOUND-Befehl schreibt die erzeugten Töne in eine Warteschlange. Ein Ton nach dem anderen wird gespielt. Erst wenn ein SOUND-Befehl beendet ist, kommt der nächste Ton dran. Das Ganze läuft übrigens wieder im Hintergrund ab: Während der Amiga Musik macht, kann sich AmigaBASIC schon wieder ganz anderen Problemen widmen. Für eine ständige Musikberieselung genügt es, hie und da ein paar neue Töne in die Warteschlange zu schreiben. Bitte nicht zu viele, denn jeder Ton kostet Speicherplatz. Wenn Sie eine ganze Partitur auf einmal runterspielen wollen, erhalten Sie schon recht früh im ersten Satz einen "Out of memory"-Error.

Wie aber kann man es erreichen, daß mehrere Stimmen gleichzeitig spielen? Wir haben ja vorhin gesagt, daß zur selben Zeit bis zu vier Töne erklingen können. Dazu brauchen Sie einen Befehl, der die Töne anhält und nicht sofort zum Spielen freigibt: SOUND WAIT. Alle nun folgenden SOUND-Befehle laufen auf, und bleiben zunächst gespeichert. Folgt dann der Befehl SOUND RESUME, wird die Warteschlange entriegelt, und alle Töne werden gespielt. Und zwar nach folgender Regel: Töne, die den gleichen Kanal benutzen, kommen hintereinander dran. Töne auf verschiedenen Kanälen gleichzeitig. Probieren wir das mal aus:

```
sound wait
sound 440,10
sound 880,10
sound resume
```

Die beiden Töne werden hintereinander gespielt, weil sie beide den Kanal 0 benutzen.

```
sound wait
sound 261.63,36,,0
sound 329.63,36,,1
sound 392,36,,2
sound resume
```

Dieser Dreiklang erklingt gleichzeitig (sollte ein Dreiklang auch...), weil jeder Ton einen anderen Kanal benutzt. Wenn Sie Musikstücke komponieren, bei denen zu unterschiedlichen Zeiten unterschiedliche Anzahlen an Tönen erklingen, müssen Sie die unbenutzen Kanäle mit 0-Lautstärken besetzen, damit die richtigen Töne beieinander bleiben. Eine Skizze verdeutlicht das:

	1. Ton	2. Ton	3. Ton	4. Ton	5. Ton	6. Ton	7. Ton
Kanal 0	523.25	784	698.48	659.28	587.28	1046.52	784
Kanal 1	659.28	987.76	0	0	0	1318.52	987.76
Kanal 2	784.00	1174.52	0	0	0	1568	1174.52
Kanal 3	-	-	-	-	-	-	-

Würden Kanal 1 und 2 beim dritten bis fünften Ton nicht mit 0-Tönen besetzt, würde der 6. Ton dieser beiden Kanäle bereits zusammen mit dem 3. Ton des ersten Kanals erklingen. Nur die Ruhe! Lesen Sie diesen Satz noch mal langsam durch und vergleichen Sie die genannten Töne mit der Skizze, dann wird's klar. Wollen Sie sich das Ergebnis mal anhören? Schreiben Sie das Programm am besten ins LIST-Window oder laden Sie es von Diskette ("StarWars3stimm").

```
SOUND WAIT

Lesen:
  READ Freq,Dauer,Lautst,Kanal
  IF Freq=-1 THEN Spielen
  SOUND Freq,Dauer,Lautst,Kanal
GOTO Lesen

Spielen:
  SOUND RESUME
```

```
DATA 523.23,15,127,0, 659.28,15,96,1, 784,15,96,2
DATA 784.15,15,222,0, 987.76,15,180,1, 1174.52,15,180,2
DATA 698.48,6,127,0, 0,6,0,1, 0,6,0,2
DATA 659.28,6,127,0, 0,6,0,1, 0,6,0,2
DATA 587.28,6,127,0, 0,6,0,1, 0,6,0,2
DATA 1046.52,15,255,0, 1318.52,15,180,1, 1568,15,180,2
DATA 784,24,180,0, 987.76,24,160,1, 1174.52,24,160,2
DATA -1,-1,-1,-1
```

Klingt gut, nicht? Die DATAs entsprechen genau der Stimmenzuteilung aus der Skizze von oben. Die 'Lesen:'-Schleife liest alle Werte für den SOUND-Befehl ein und SOUND schreibt sie in die Noten-Warteschlange. Wenn als Frequenz der Wert -1 eingelesen wurde, erkennt das Programm, daß nun alle DATAs beendet sind. Sollte Ihnen die Stimmlage der Melodie zum Mitsingen oder Genießen zu hoch sein, teilen Sie die Frequenz nach dem Lesen einfach durch 2:

```
SOUND Freq/2,Dauer,Lautst,Kanal
```

Schon spielt sich die ganze Sache eine Oktave tiefer ab. Von noch tieferen oder höheren Frequenzen raten wir jedoch ab.

Mit diesen Befehlen steht Ihnen jetzt schon eine recht große Spielwiese für eigene Kompositionen zur Verfügung. Aber es kommt noch toller.

7.4 Horchet, wie es klinget - ein bißchen Ton-Theorie

Bevor wir die weiteren Möglichkeiten der Musikerzeugung in AmigaBASIC besprechen, sollten Sie ein paar technische Grundlagen erfahren, damit Sie das, was wir danach erklären, besser und leichter verstehen.

Daß jeder Ton letztlich eine Luftschwingung ist, wissen Sie schon. Jedes Gerät, das auf irgendeine Art Musik erzeugt oder wiedergibt, muß also eine Schwingung produzieren. Ein Musikinstrument überträgt die Schwingung einer Saite (Gitarre, Geige,

Klavier) oder Lufterschütterungen (Trompete, Trommeln) auf die Umgebung. Stereoanlagen und Computer verwenden die Membran eines Lautsprechers zur Schwingungserzeugung. Zur Übermittlung der Tonsignale sendet der Verstärker an den Lautsprecher ein Stromsignal, das abhängig von der Schwingung stärker oder schwächer wird. Die Eigenschaften eines Tons hängen von der Beschaffenheit der Schwingung ab. Um sich Luftschwingungen besser vorstellen zu können, vergleicht man sie mit Wellen: Ähnlich wie eine Wasseroberfläche auf und ab schwingt, bewegen sich auch die Luftschichten.

Eine typische Wellenform für Töne ist die Sinuskurve. Ja richtig - genau die Sinuskurve, die uns schon im Grafik-Kapitel tatkräftig unterstützt hat. Spricht man von einer "Sinuswelle" und meint dabei Töne, bedeutet das, daß die Schwingung wie die Werte der Sinusfunktion gleichmäßig an- und absteigt. Allerdings ziemlich schnell. Für unseren Kammerton a zum Beispiel 440mal pro Sekunde. Schauen Sie sich mal Bild 19 an. Diese Sinuskurve steht für eine komplette Schwingung: Die Welle beginnt auf Höhe der Mittellinie. Sie fällt ab, steigt wieder an, durchquert die Mittellinie, steigt weiter an, fällt wieder ab, und in dem Augenblick, in dem sie wieder die Mittellinie erreicht, ist die Schwingung beendet. (Auch wenn es sich so anhört: Das ist keine Übertragung eines Fußballänderspiels, sondern Tontheorie...) 440mal in einer Sekunde muß diese Schwingung passieren, damit wir ein 'a' hören. Die Frequenz gibt die Anzahl der Schwingungen pro Sekunde an. Je höher dieser Wert, um so höher hören wir den Ton.

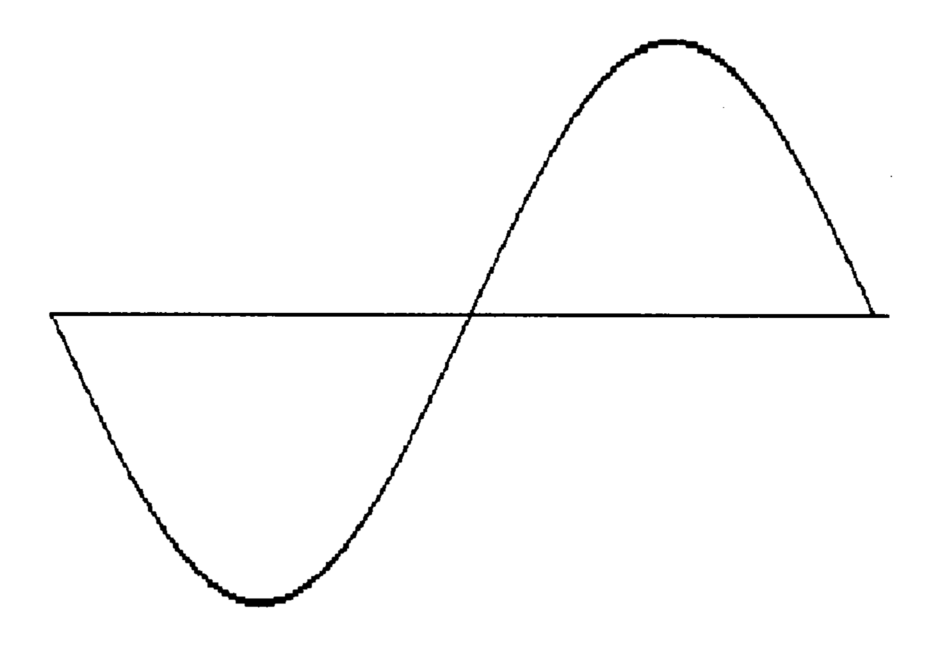


Bild 19: **Eine Sinusschwingung**

Unterhalb einer bestimmten Grenze, sie liegt in der Gegend von 20 Hertz, empfinden wir eine Schwingung nicht mehr als Ton. Das ist auch gut so, sonst würden wir jeden Luftzug und jede Luftbewegung hören. Nur wenn die Schwingung sehr groß ist, hören wir doch etwas. Zum Beispiel den Knall, wenn jemand eine Tür zuschlägt. Die Höhe eines Tons hängt also von der Frequenz der Schwingung ab.

Auch die Lautstärke hängt von der Schwingung ab: Je stärker eine Welle schwingt, umso lauter hören wir sie. Die Stärke einer Schwingung, also der Abstand des höchsten Punkts der Welle zur Grundlinie nennt der Fachmann "Amplitude". Die Amplitude der Schwingung ist für den Lautstärkeverlauf des Tons zuständig.

Auch andere Eigenschaften der Welle beeinflussen den Ton: Für die Klangfärbung, also den typischen Klang eines Instruments oder eines Geräuschs, ist die Form der Welle ausschlaggebend. Darüber sprechen wir ausführlich im nächsten Kapitel.

Um eine Tonwelle zu speichern und später wiederzugeben, benutzte man bis vor kurzem ausschließlich die sogenannte "analoge" Tonaufzeichnung: Beim Abspielen einer Schallplatte wird die Abtastnadel in Schwingungen versetzt, die dann an den Verstärker weitergegeben werden. Bei Tonbändern und Cassetten liest der Tonkopf magnetische Signale, deren Stärke sich ändert. Seit einigen Jahren gibt es aber ein anderes, moderneres Konzept, das seinen Weg in Form von CDs (Compact Disks) und Synthesizern auch in die Privathaushalte gefunden hat: Die digitale Tonaufzeichnung. "Digital" hat mit Computern und Bits zu tun. Wen wundert's dann, daß dieses Konzept auch im Amiga benutzt wird. Wie es funktioniert? Nun, was kann sich der Amiga besser merken als Zahlen. Bei digitaler Tonerzeugung, -speicherung und -wiedergabe werden analoge Schwingungen durch digitale Werte vertreten.

Das Prinzip ist eigentlich ganz einfach. Schauen wir uns zunächst die digitale Tonaufzeichung bzw. -speicherung an, wie sie zum Beispiel bei der Aufnahme von CDs verwendet wird: Mehrmals pro Sekunde wird der aktuelle Stand der Schwingung abgetastet und in eine Zahl umgerechnet. Je höher die Amplitude, desto größer die Zahl. Findet die Abtastung häufig genug statt, entsteht eine Reihe von Zahlen, die den Verlauf der Welle recht gut wiedergibt. Und zwar sowohl die Frequenz (Tonhöhe) als auch die Amplitudengröße (Lautstärke) und die Wellenform der Schwingung.

Das Verfahren, Schwingungen durch Zahlen zu repräsentieren, heißt Sampling. ("Sample" bedeutet auf Deutsch "Probe, Stichprobe". Sampling ist das Erstellen von Stichproben.) Wie realistisch ein durch Sampling erzeugter Ton klingt, hängt in erster Linie von der Sampling-Rate ab. Die Sampling-Rate, also die

Anzahl an Amplituden-Abfragen, kann beim Amiga bis zu 30000 Werte pro Sekunde betragen. Ein CD-Plattenspieler tastet mit über 40000 Werten pro Sekunde ab.

Bei der Wiedergabe oder der künstlichen Erzeugung von Musik läuft der Vorgang genau umgekehrt ab: Der Chip "Paula", der unter anderem für die Tonerzeugung zuständig ist, wandelt die digitalen Werte, die irgendwo im Speicher stehen, in analoge Stromsignale um. Dazu gibt es sogenannte Digital/Analog-Wandler. Das sind kleine elektronische Bauteile, die eine digitaliserte Zahl (also z.B. ein Byte) einlesen und je nach Größe der Zahl einen entsprechend starken Stromwert erzeugen. Dieser Strom muß dann nur noch zum Tonausgang des Amiga gelenkt werden, schon ist ein digitaler Sound erzeugt. Und weil in Paula vier D/A-Wandler eingebaut sind, stehen uns eben auch vier Tonkanäle zur Verfügung.

7.5 Plätscher, Plätscher - Wellenformen

Welche Eigenschaften einer Welle für Lautstärke und Tonhöhe verantwortlich sind, wissen Sie mittlerweile: Die Frequenz steuert die Tonhöhe, die Amplitude ist für die Lautstärke zuständig. Fast noch wichtiger als diese beiden Werte ist die Wellenform. Sie bestimmt den Klang des Tons. Wenn man eine Tonwelle, die von einem Musikinstrument oder auf irgendeine andere Weise erzeugt wurde, grafisch darstellt, sieht man für jeden Ton-Verursacher sehr charakteristische Wellenformen. Eine Sinuswelle klingt sehr weich, sehr harmonisch. Reine Sinusschwingungen kommen aber in der Natur nicht vor. Schlaginstrumente (wie zum Beispiel eine Trommel) erzeugen unregelmäßige, eckige, scharfe Wellenformen.

Je höher die Sampling-Rate ist, desto besser kann der Verlauf der Original-Welle nachgebildet werden und desto realistischer klingt der Ton. Vielleicht haben Sie schon Programme für Ihren Amiga gesehen (bzw. gehört), die kurze Musikpassagen in erstaunlich guter Qualität wiedergeben. Solche Programme benutzen eine sehr hohe Samplingrate und spielen nur eine kurze Passage eines digitalisierten Musikstücks. Meist reicht schon

nach wenigen Sekunden der Speicherplatz nicht mehr aus: Selbst 512 KByte sind in dieser Hinsicht recht wenig. Eine Compact Disk, die ca. 70 Minuten akustisch hochwertige Musik speichern kann, hat eine Speicherkapazität von 550 MegaByte. Trotz dieser Einschränkungen sind die Möglichkeiten des Amiga auf dem Gebiet der Musikerzeugung beachtlich. AmigaBASIC andererseits ist recht zurückhaltend, was die Unterstützung dieser Möglichkeiten betrifft. Es gibt nur zwei Befehle zur Tonerzeugung: SOUND und WAVE. Den SOUND-Befehl kennen Sie schon. Und WAVE ermöglicht die Verwendung eigener Wellenformen.

Die Wellenform, die von AmigaBASIC normalerweise zur Tonerzeugung verwendet wird, ist eine Sinuswelle. In allen unseren Beispielen hörten wir bisher nur diese Wellenform. Wollen Sie eine andere Wellenform benutzen, geben Sie einfach ein Integerfeld an, das mindestens 256 Elemente enthält und in dem die gesampelten Wellen stehen. Die Zahlen im Feld dürfen zwischen -128 und 127 liegen. Das entspricht einer Analog/Digital-Auflösung von 8 Bit: Der digitale Wert, der bei der Umwandlung herauskommt, ist eine 8-Bit-Zahl.

Zum Vergleich: Ein CD-Plattenspieler bietet eine Auflösung von 16 Bits. Das heißt aber nicht, daß so ein CD-Player doppelt so gute Ergebnisse liefert wie der Amiga: Durch verschiedene Tricks in Hard- und Software klingt die Amiga-Tonerzeugung wesentlich besser als man aufgrund der technischen Daten erwarten würde.

Aber zurück zu WAVE: Der Wert -128 im Feld bedeutet, daß die Welle zum gegenwärtigen Zeitpunkt den tiefsten möglichen Wert unterhalb der Grundlinie hat. Und 127 ist der größte positive Wert für die Schwingung. Wir können nun ein Programm Werte für das Integerfeld ausrechnen lassen und so eigene Wellenformen erzeugen, die AmigaBASIC zur Tonwiedergabe verwendet. Stellt sich bloß die Frage, wie so eine Wellenform aussehen soll.

Nun, es gibt neben Sinuswellen noch weitere bekannte und beliebte Wellenformen, die recht unterschiedliche Ergebnisse haben. Zuerst wäre da die Dreiecks-Schwingung. Sie verläuft ähn-

lich wie die Sinuswelle, klingt aber schärfer, weil die Schwingung nicht langsam ihre Werte ändert, sondern sehr abrupt umkehrt. Eine grafische Darstellung dieser Wellenform sehen Sie im Bild 20.

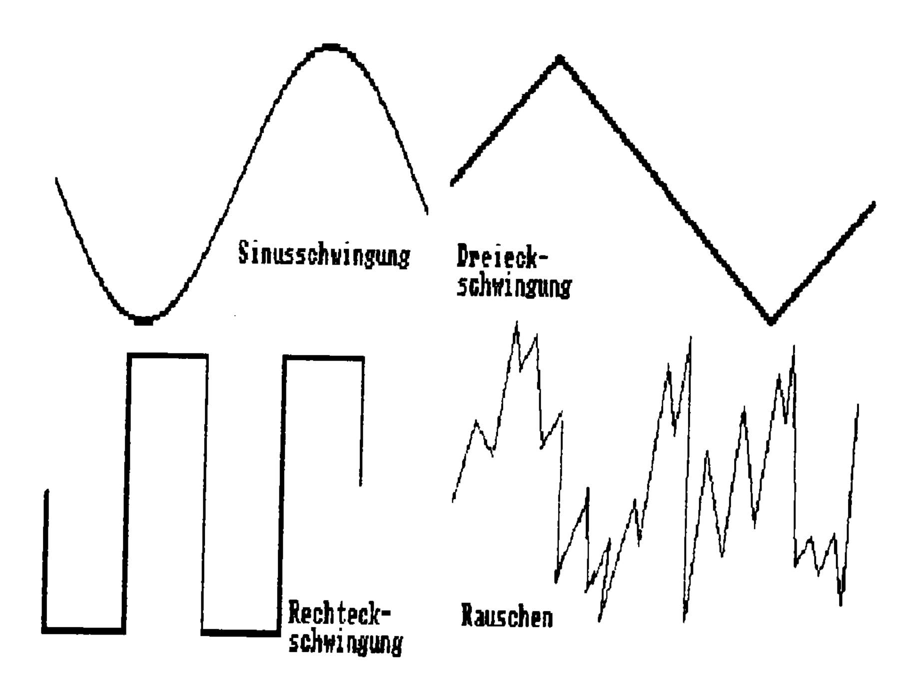


Bild 20: Verschiedene Wellenformen

Wir wollen das Ganze anhand unseres dreistimmigen StarWars-Musikprogramms ausprobieren. Laden Sie es bitte und geben Sie den folgenden Programmteil ganz am Anfang ein. Auf der Diskette hat diese Komposition den Namen "SW.Dreieck".

```
DIM Dreieck%(256)
FOR x=0 TO 62
  a=a+2
  Dreieck%(x)=a
NEXT x
```

```
FOR x=63 TO 188
  a=a-2
  Dreieck%(x)=a
NEXT x
FOR x=189 TO 255
  a=a+2
  Dreieck%(x)=a
NEXT x

WAVE 0,Dreieck%
WAVE 1,Dreieck%
WAVE 2,Dreieck%

ERASE Dreieck%
```

Das Integerfeld 'Dreieck%' muß zunächst dimensioniert werden. Die erste FOR...NEXT-Schleife erzeugt die Feldinhalte für den Teil der Dreieckschwingung, der von der Grundlinie zum ersten (positiven) Extremwert reicht. Die zweite Schleife berechnet die Werte für den Rückfall auf den untersten Extremwert. Und das dritte FOR...NEXT sorgt für den Anstieg zurück bis zur Grundlinie. Vergleichen Sie diese Phasen der Dreieckswelle bitte mit der Abbildung in Bild 20.

Mit dem Befehl WAVE wird das Feld 'Dreieck%' dann der Reihe nach den drei verwendeten Stimmen zugewiesen. Sie können aber auch auf jeder der vier Amiga-Stimmen eine andere Wellenform benutzen. Die Zuweisung mit WAVE reserviert jeder Stimme einen Speicherbereich, aus dem sie ihre Sampling-Werte lesen kann. Deshalb hat das Feld 'Dreieck%' nach der Zuweisung mit WAVE jede Existenzberechtigung verloren, wir löschen es mit ERASE. Bei großen oder mehreren Wellenform-Dateien ist das wirklich ratsam, denn sonst würde viel Speicherplatz sinnlos verschwendet.

Wenn das Programm läuft, werden Sie hören, daß sich die Töne jetzt etwas anders anhören. Die Wellenform wird auch im Direktmodus weiterverwendet. So können Sie im BASIC-Window

noch zusätzliche Experimente machen. Beachten Sie bitte, daß sogar der BEEP sich anders anhört als vorher - auch er läuft über Tonkanal 0.

Wenn Sie sich jetzt nach Ihrem guten alten, vertrauten BEEP sehnen, haben wir eine gute Nachricht für Sie. Geben Sie doch mal ein:

```
WAVE 0,SIN
```

Durch diesen Befehl wird die Wellenform von Kanal 0 auf die Sinusschwingung zurückgesetzt. SIN ist kein Feld, sondern eine Option des WAVE-Befehls, um schnell wieder zur Grundeinstellung (eben der Sinuswelle) zurückzukommen.

Die nächste Wellenform, die wir Ihnen vorstellen wollen, heißt Rechteckschwingung. Sie besteht nur aus Sprüngen von einer Amplitude zur anderen. Der so erzeugte Klang ist etwas hölzern. Eine grafische Darstellung finden Sie wieder im Bild 20. Und wenn Sie's mal hören möchten, speichern Sie Ihre Dreiecks-Version bitte ab und verbessern Sie den Anfangsteil des Programms, so daß das folgende Listing dabei herauskommt. Oder laden Sie schlicht und einfach "SW.Rechteck".

```
DIM Rechteck%(256)
FOR x=0 TO 127
  Rechteck%(x)=127
NEXT X
FOR X=128 TO 256
  Rechteck%(x)=-128
NEXT X

WAVE 0,Rechteck%
WAVE 1,Rechteck%
WAVE 2,Rechteck%

ERASE Rechteck%
   .
   .
   .
```

Der restliche Teil bleibt unverändert. Eine Rechteckwelle ist sehr einfach zu erzeugen: In die erste Hälfte des Datenfelds 'Rechteck%' lesen wir den Maximalwert 127 und in die zweite Hälfte lesen wir -128 ein, den tiefstmöglichen Wert. Der ständige Wechsel zwischen diesen beiden Amplituden erzeugt die gewünschte Schwingung. Die Abbildung in Bild 20 zeigt natürlich mehrere Rechteckschwingungen, nicht nur eine.

Wenn Sie dieses Programm abspeichern, geben Sie ihm bitte einen neuen Namen, vielleicht "Rechteck".

Als dritten und letzten im Bunde stellen wir Ihnen noch die Wellenform "Rauschen" vor. Diese Welle folgt überhaupt keinen festen Regeln, sie ist vielmehr eine Zufallsschwingung. Dabei kommen Töne heraus, die an eine elektrische Klingel oder einen Wecker erinnern. Bei tiefen Frequenzen können Sie damit auch Explosionen oder ähnliche Geräusch-Effekte erzielen. In unserem Bild 20 sehen Sie auch eine grafische Darstellung einer typischen "Rauschen"-Wellenform.

```
DIM Rauschen%(256)
FOR x=0 TO 256
  Rauschen%(x)=RND*255-128
NEXT x

WAVE 0,Rauschen%
WAVE 1,Rauschen%
WAVE 2,Rauschen%

ERASE Rauschen%
```

Die Formel RND*255-128 erzeugt Zufallszahlen zwischen -128 und 127. Das ist genau der Bereich, den wir für unser Feld 'Rauschen%' brauchen. Auf der beiliegenden Diskette finden Sie dieses Beispiel unter "SW.Rauschen".

Jetzt haben Sie Beispiele für drei verschiedene selbst erzeugte Wellenformen kennengelernt. Die Dreieckswelle klingt etwas schärfer, transparenter als die Sinuswelle. Und die Rechteckwelle klingt hölzern, aber klar. Interessanterweise sind tiefe Töne, die in den Wellenformen "Rechteck" und "Rauschen" gespielt werden, besser zu hören als dieselben Frequenzen bei einer Sinuswelle oder einer Dreieckswelle.

Sicher möchten Sie jetzt verschiedene, eigene Wellenformen ausprobieren. Um Ihnen dieses Ausprobieren zu erleichtern, haben wir mal wieder ein Utility für Sie geschrieben. Es ist auch gleichzeitig das letzte Utility, das wir in diesem Buch zusammen schreiben werden. Na, na, Sie werden doch nicht schon in Abschiedsstimmung geraten? Das wäre dann doch noch etwas verfrüht.

7.6 Ausklang - das Synthesizer-Utility

Unser Synthesizer-Programm ist mit Sicherheit kein Ersatz für Musikprogramme wie "Musicraft", "Instant Music" oder das "Deluxe Music Construction Set". Töne zu erzeugen, die auch nur ansatzweise mit den Ergebnissen solcher Programme vergleichbar wären, ist mit den eingeschränkten musikalischen Fähigkeiten von AmigaBASIC unmöglich.

Das folgende Programm ist dafür gedacht, eigene Wellenformen zu erzeugen. Sie können sie ausprobieren, und wenn Sie einen schönen Klang gefunden haben, das WAVE-Datenfeld abspeichern und später in Ihren eigenen Programmen benutzen. Das Programm finden Sie unter dem Namen "Synthesizer" in der "Musik"-Schublade unserer Diskette. Sie können es natürlich auch selbst abtippen.

```
Vorbereitungen:
  DIM Welle%(256)
  DEF FNYWelle%(a)=ABS(Welle%(a)-128)
  SCREEN 1,320,200,2,1
  WINDOW 2,"Wellenform",(0,0)-(256,63),22,1
```

```
  FOR x=0 TO 256
    Welle%(x)=127*SIN(x/20)
  NEXT x

  WINDOW 3,"Funktionen",(195,80)-(310,175),22,1
  WINDOW OUTPUT 3

  LINE (5,5)-(55,30),1,b
  PSET (5,17)
  FOR x=0 TO 48
    LINE -((x+5),17-10*SIN(x/3.8))
  NEXT x

  LINE (59,5)-(110,30),1,b
  LINE (59,18)-(67,7) : LINE -(83,27)
  LINE -(99,7) : LINE -(107,18)
  LINE (5,35)-(55,60),1,b
  LINE (7,47)-(7,37)
  LINE -(18,37) : LINE -(18,57)
  LINE -(30,57) : LINE -(30,37)
  LINE -(41,37) : LINE -(41,57)
  LINE -(53,57) : LINE -(53,47)
  LINE (59,35)-(110,60),1,b
  LOCATE 6,9 : PRINT "Lösch"
  LINE (5,65)-(55,90),1,b
  LOCATE 10,2 : PRINT "Save"
  LINE (59,65)-(110,90),1,b
  LOCATE 10,9 : PRINT "Load"

  GOSUB WelleZeigen

  ON MOUSE GOSUB Maussteuerung
  MOUSE ON

  WINDOW 3

Eingabe:
  a$=INKEY$
  f=0
  IF a$="" THEN Eingabe
```

```
    IF a$=CHR$(9) THEN f=261.63
    IF a$="1" THEN f=277.18
    IF a$="q" THEN f=293.66
    IF a$="2" THEN f=311.13
    IF a$="w" THEN f=329.63
    IF a$="e" THEN f=349.23
    IF a$="4" THEN f=369.99
    IF a$="r" THEN f=392.00
    IF a$="5" THEN f=415.3
    IF a$="t" THEN f=440.00
    IF a$="6" THEN f=466.16
    IF a$="z" THEN f=493.88
    IF a$="u" THEN f=523.25
    IF a$="8" THEN f=554.37
    IF a$="i" THEN f=587.58
    IF a$="9" THEN f=622.25
    IF a$="o" THEN f=659.28
    IF a$="p" THEN f=698.48
    IF a$="ß" THEN f=739.99
    IF a$="ü" THEN f=784
    IF a$="'" THEN f=830.61
    IF a$="+" THEN f=880.00
    IF a$="\" THEN f=932.33
    IF a$=CHR$(13) THEN f=987.76
    IF a$=CHR$(8) THEN f=1046.52
    IF f=0 THEN Eingabe

  Spielen:
    Laut=127 : if f=0 then l=0
    SOUND WAIT
    SOUND f,3,Laut,0
    SOUND f,3,Laut,1
    SOUND RESUME
  GOTO Einlesen

  Maussteuerung:
    IF WINDOW(0)=2 THEN WellenformAendern
    IF WINDOW(0)=3 THEN FunktionAendern
```

```
WellenformAendern:
  WINDOW 2
  WHILE MOUSE(0)<0
    x=MOUSE(5)
    IF x>256 THEN GOSUB WelleZeigen : RETURN
    IF x<1 THEN x=1
    y=MOUSE(6)
    IF y>63 THEN GOSUB WelleZeigen : RETURN
    LINE (x-1,FNYWelle(x-1)/4)-(x,FNYWelle(x)/4),0
    LINE (x-1,FNYWelle(x-1)/4)-(x,y),1
    Welle%(x)=127-(y*4)
  WEND
  GOSUB WelleZeigen
RETURN

FunktionAendern:
  Test=MOUSE(0)
  x=MOUSE(3)
  y=MOUSE(4)
  IF x>4 AND x>56 AND y>4 AND y<31 THEN
    WINDOW 3 : PAINT (7,6),3,1
    FOR x=0 TO 256
      Welle%(x)=127*SIN(x/20)
    NEXT x
    GOSUB WelleZeigen
    WINDOW 3 : PAINT (6,6),0,1
  END IF
  IF x>58 AND x<111 AND y>4 AND y<31 THEN
    WINDOW 3 : PAINT (60,6),3,1
    FOR x=0 TO 256
      IF x<41 then Welle%(x)=x*3 : a=x*3
      IF (x>=41 AND x<126) OR (x>=210) THEN a=a-2.57 : Welle(x)=a
      IF x>=126 AND y<210 THEN a=a+2.57 : Welle%(x)=a
    NEXT x
    GOSUB WelleZeigen
    WINDOW 3 : PAINT (60,6),0,1
  END IF
  IF x>4 AND x<61 AND y>34 AND y<61 THEN
    WINDOW 3 : PAINT (6,36),3,1
    FOR x=0 TO 256
```

```
      IF x<64 OR (x>=128 AND x<191) THEN Welle%(x)=127
      IF (x>=64 AND x<128) OR x>192 THEN Welle%(x)=-128
    NEXT x
    GOSUB Wellezeigen
    WINDOW 3 : PAINT (6,36),0,1
  END IF
  IF x>58 AND x<111 AND y>34 AND y<61 THEN
    WINDOW 3 : PAINT (60,36),3,1
    FOR x=0 TO 256
      Welle%(x)=0
    NEXT x
    GOSUB WelleZeigen
    WINDOW 3 : PAINT (60,36),0,1
  END IF
  IF x>4 AND x<61 AND y>64 AND y<91 THEN
    WINDOW 3 : PAINT (6,66),3,1
    GOSUB EingabeName
      IF Nam$="" THEN PAINT (6,66),0,1 : RETURN
      OPEN Nam$ FOR OUTPUT AS 1
        FOR x=0 TO 256
          PRINT #1,CHR$(127-Welle%(x));
        NEXT x
      CLOSE 1
      WINDOW 3 : PAINT (6,66),0,1
  END IF
  IF x>58 AND x<111 AND y>64 AND y<91 THEN
    WINDOW 3 : PAINT (62,66),3,1
    GOSUB EingabeName
      IF Nam$="" THEN PAINT (62,66),0,1 : RETURN
      OPEN Nam$ FOR INPUT AS 1
        FOR x=0 TO 256
          Welle%(x)=127-ASC(INPUT$(1,1))
        NEXT x
      CLOSE 1
      WINDOW 3 : PAINT (62,66),0,1
  END IF
RETURN

WelleZeigen:
  WINDOW 2 : CLS
```

```
    FOR x=1 TO 256
      LINE (x-1,FNYWelle(x-1)/4)-(x,FNYWelle(x)/4),1
    NEXT x
  WINDOW 3
  WAVE 0,Welle%
  WAVE 1,Welle%
RETURN

EingabeName:
  WINDOW 4,"Bitte Dateinamen eingeben:",(5,100)-(300,110),0,1
  CLS : LINE INPUT Nam$
  IF Nam$= "=" OR Nam$= "*" THEN Nam$=Altname$
  IF Nam$<>"" THEN Altnam3$=Nam$
  WINDOW CLOSE 4 : WINDOW 3
RETURN
```

Wenn Sie es tatsächlich selbst abgetippt haben, speichern Sie das Programm bitte gleich ab.

Dann gehen wir daran, die einzelnen Programmfunktionen zu besprechen. In unseren bekannten und beliebten 'Vorbereitungen:' wird zunächst das Integerfeld 'Welle%' dimensioniert. Wir verwenden 257 Einzelwerte für unsere Welle.

Der nächste Befehl ist gleich wieder ein Unbekannter: DEF FN. Wir haben ihn bisher noch nicht gebraucht, aber in Programmen, die viele Berechnungen durchführen müssen, ist er sehr hilfreich. DEF FN ("Define Function" - "Definiere eine Funktion") dient dazu, eigene BASIC-Funktionen festzulegen. y=SIN(x) ist eine Funktion, die fest in AmigaBASIC eingebaut ist. Wenn Sie aber zum Beispiel innerhalb eines Programms oft die Funktion y=SIN(x)*COS(x)+0.5*(SIN(x)-COS(x)) benötigen, wäre es unzweckmäßig und unübersichtlich, jedesmal den ganzen langen Ausdruck einzutippen. Sie können sich stattdessen mit DEF FN eine eigene Funktion definieren. Für unser Beispiel müßte am Beginn des Programms stehen: DEF FNHallo(x)=SIN(x)*COS(x)+0.5*(SIN(x)-COS(x)). Von da an können Sie im Programm die Funktion 'FNHallo(x)' verwenden. Der Name der Funktion ist 'Hallo'. Um so ein 'Hallo' von nor-

malen Variablen unterscheiden zu können, steht das FN davor. Wenn Sie PRINT FNHallo(1) schreiben, sieht AmigaBASIC in der Funktionsdefinition nach und rechnet die eingegebene Formel für den Wert 1 aus.

In der Funktionsdefinition geben Sie hinter dem Funktionsnamen eine oder mehrere Variablen an. Dann kommt die Formel, die mit den Werten dieser Variablen durchgeführt werden soll. Den Variablennamen in der Definitionszeile können Sie im Programm normal verwenden, er dient bloß dazu, die Funktion festzulegen. Noch nicht ganz verstanden?

Nehmen wir ein ganz einfaches Beispiel: DEF FNVerdopple(x) = 2*x. Wenn irgendwo im Programm 'FNVerdopple(2)' steht, schaut AmigaBASIC in der DEF FN-Zeile nach der Definion von 'FNVerdopple' und weiß dann: "Wenn hinter 'FNVerdopple' ein Wert steht wie das x in der DEF FN-Zeile, muß ich diesen Wert verdoppeln wie das x in der DEF FN-Zeile." Geben Sie eine Funktion an, die nicht mit DEF FN definiert wurde, erhalten Sie einen "Undefined user function"-Error: "Nicht definierte Benutzer-Funktion."

In unserem Synthesizer-Programm benutzen wir die Funktion 'FNYWelle'. Sie rechnet die Werte, die wir zur Felddefinition verwenden, in Werte zwischen 0 und 255 um. Zur Erinnerung: WAVE-Felder müssen Werte zwischen -128 und +127 beinhalten. Wenn wir von so einem Wert 128 abziehen und das Vorzeichen weglassen, bekommen wir eine Zahl zwischen 0 und 255. Auf BASIC: DEF FNYWelle(a)=ABS(Well%(a)-128).

Unser Programm läuft auf einem niedrigauflösenden Screen (320 mal 200 Punkte). Dort erzeugen wir zunächst ein Window, das Window 2 ("Wellenform"), in dem Sie die Wellenform festlegen können.

Als Grundeinstellung benutzt das Synthesizer-Programm eine Sinuswelle. Sie unterscheidet sich aber etwas von der Sinuswelle, die AmigaBASIC normalerweise benutzt. Die Werte, die für unsere Sinuswelle in 'Welle%' stehen müssen, errechnet die FOR...NEXT-Schleife im 'Vorbereitungen:'-Teil.

Das Window beinhaltet die Kästchen, in die Sie für bestimmte Funktionen klicken müssen. Es gibt ein Kästchen, um eine Sinuswelle als Wellenform zu übernehmen, ein Kästchen für eine Dreieckswelle und ein Kästchen für eine Rechtecks-Welle. Wie diese Wellen aussehen müssen, wissen Sie ja schon. Wir erzeugen die Symbole dafür mit LINE-Befehlen. Für "Sinus" wird im ersten Kästchen eine kleine Sinuskurve gezeichnet. Für "Dreieck" malen wir aus vier LINE-Befehlen eine Dreieckwelle. Bei "Rechteck" brauchen wir insgesamt neun Linien. Außer diesen Kästchen gibt es noch ein Kästchen für Löschen der Welle, eines zum Speichern und eines zum Lesen von Wellendaten.

Ist der Aufbau des Windows 3 erledigt, rufen wir 'WelleZeigen:' auf, das im Window 2 die aktuelle Wellenform (am Programmanfang eben eine Sinuswelle) darstellt. Das Ausgabewindow für unser Programm ist bei allen anderen Funktionen Window 3. Während Sie Töne spielen, müssen Sie dieses Window aktiviert haben. AmigaBASIC kann Tastatureingaben ja immer nur einem Window zuordnen, und dafür haben wir eben Window 3 ausgewählt. Vorher wird übrigens noch Event Trapping für die Maus aktiviert.

Der Programmteil "Eingabe:" checkt die Tastatureingaben ab. Wir haben in der obersten und zweiten Tastaturreihe eine Klaviatur simuliert. Die Tasten <TAB>, <Q>, <W>, <E>, und so weiter bis <+>, <RETURN> und <BACKSPACE> erzeugen die Grundtöne, die darüberliegenden Tasten entsprechen den schwarzen Tasten auf dem Klavier. Abhängig von 'a$', dem eingegeben String, weisen wir 'f' eine Frequenz zu. Die Frequenzen der Grundtöne kennen Sie ja aus Tabelle 12. Die Frequenzen der Zwischentöne haben wir zusätzlich ausgerechnet, Sie sehen die Werte im Programmlisting.

Aus Werten wie 440.00 macht AmigaBASIC ein 440!. Aber das ist mit Ihrem Wissen über einfach genaue Fließkommazahlen (das stammt aus Zwischenspiel 7, unserem Gruselkabinett - wer's nicht mehr weiß, muß wieder zurück!) sicher keine Überraschung für Sie.

Der Programmteil 'Spielen:' spielt den Ton zur gedrückten Taste. Wir schicken jeweils einen Ton auf den linken und einen auf den rechten Kanal. Beide Töne starten dann gleichzeitig mit SOUND RESUME. So kommt der Ton später aus beiden Lautsprechern, wenn Sie Ihren Amiga stereo hören.

In 'Maussteuerung:', dem Label, das durch Event Trapping bei einem Mausklick angesprungen wird, verzweigt das Programm zu den Unterprogrammen 'WellenformAendern:' und 'FunktionAendern:', abhängig vom Window, in das geklickt wurde. Die Funktion WINDOW(0) liefert uns die Windownummer.

In 'WellenformAendern:" wird Window 2 zum Ausgabewindow gemacht. In diesem Window können Sie mit der Maus eine Wellenform eingeben, ähnlich wie Sie auch im Malprogramm arbeiten. Der Wellenteil, der vorher an der aktuellen Stelle war, wird gelöscht. Wenn Sie mit der Maus aus dem Window fahren (IF x>256... oder IF y>63...), rufen wir das Unterprogramm 'WelleZeigen:' auf, das die passende Wellenform im Window 2 neu aufbaut, um die nicht gelöschten Reste der alten Welle vom Bildschirm zu bekommen, und springen danach mit RETURN zurück. Die dann im Programm folgenden LINE-Befehle dienen zum Löschen der alten Wellenteile (Farbe 0) und zum Zeichnen der neuen (Farbe 1). Zuletzt bekommt die Position in 'Welle%', die ja identisch mit der x-Koordinate des Mausklicks ist, ihren neuen Wert. Die Variable 'y', die y-Koordinate des Mausklick-Punkts, liegt zwischen 0 und 63 (der Windowhöhe). Um daraus Werte zwischen -128 und +127 zu bekommen, benutzen wir die Formel 127-(y*4).

Zuletzt wird in jedem Fall das Unterprogramm 'WelleZeigen:' aufgerufen. Manchmal - vor allem bei schnellen Bewegungen mit der Maus - bleiben nämlich Teile der alten Welle auf dem Bildschirm stehen, die gar nicht mehr im Feld 'Welle:' stehen. Deshalb zeichnen wir die Wellenform nach jeder Änderung neu. Außerdem wird in 'WelleZeigen:' der Feldinhalt mit WAVE den beiden Stimmen zugeordnet.

Jetzt kommt der Teil 'FunktionAendern:'. Er ist für die Auswertung und Bearbeitung der Klicks von Window 3 (mit dem Titel "Funktionen") zuständig. Die Variablen 'x' und 'y' bekommen die Startkoordinaten der Mausbewegung zugewiesen. So erhalten wir in jedem Fall den Klickpunkt und nicht die aktuellen Koordinaten, die ja unter Umständen an ganz anderer Stelle liegen können. (Dann nämlich, wenn der Anwender nach dem Klick die Maus schnell wegbewegt.)

Der erste IF/END IF-Block checkt ab, ob ins "Sinus"-Kästchen geklickt wurde. Falls ja, schreiben wir die Werte einer Sinuswelle ins 'Welle%'-Feld, rufen 'WelleZeigen:' auf und springen zurück. Zur Bestätigung des Mausklicks wird übrigens immer am Anfang der Bearbeitung das angeklickte Kästchen ausgemalt und vor dem Rücksprung wieder gelöscht. Der nächste Teil funktioniert fürs nächste Kästchen ("Dreieck") genauso, nur wird eben eine Dreieckwelle erzeugt. Dazu wird die Variable 'a' abwechselnd nach oben und nach unten gezählt: Wir erhalten die Werte für eine Dreiecksschwingung. Auch der Programmteil, der fürs "Rechteck"-Kästchen zuständig ist, funktioniert so. Er erzeugt eine Rechteckwelle. Die Feldinhalte von 'Welle%' bekommen eine Zeit lang den Wert 127, dann -128, dann wieder 127 und so weiter.

Mit dem Kästchen "Löschen" löschen Sie die aktuelle Wellenform. Dazu werden einfach alle Werte von 'Welle%' auf 0 gesetzt.

Bleiben noch die Kästchen 'Save' und 'Load'. Im ersten Fall wird eine sequentielle Datei zum Schreiben geöffnet, im zweiten zum Lesen. Der Programmteil 'EingabeName:' - ein guter alter Bekannter aus vielen Programmen dieses Buchs - sorgt vorher für den Dateinamen. Und schon ist auch 'FunktionAendern:' vorbei. Jetzt fehlen nur noch die beiden Unterprogramme.

'WelleZeigen:' stellt die aktuelle Welle im Window 2 dar. Für die Berechnung der Bildschirmkoordinaten können wir gut unsere Funktion 'FNYWelle' gebrauchen. Sie kam übrigens auch schon beim 'WellenformAendern:' zum Einsatz. Aus den Ergebnissen im Bereich 0 bis 256 errechnen wir mit Division durch 4 die

Bildschirmkoordinaten. Die beiden WAVE-Befehle am Ende bewirken, daß die Wellenformen dann auch wirklich für die Tonerzeugung benutzt werden.

'Eingabename:' verlangt die Eingabe eines Dateinamens. Sie können den zuletzt verwendeten wieder mit "=" oder "*" abkürzen. Zu Eingabe erzeugen wir kurzzeitig übrigens ein viertes Window in der Bildschirmmitte.

Tja, und das war's dann auch. Die Bedienung des Programms ist ganz einfach. Malen Sie mit der Maus ins "Wellenform"-Window eine Wellenform. Sie werden schon bald Erfahrung darin haben, welche Wellenform wie klingt. Auf den oberen beiden Tastenreihen können Sie dann Töne spielen. Leider ist nicht mehr als ein Ton gleichzeitig möglich, weil nicht mehr Tasten gleichzeitig abgefragt werden können.

Wollen Sie zuerst eine der Grundformen übernehmen, um die dann abzuändern, klicken Sie einfach ins entsprechende Kästchen im "Funktionen"-Window. Mit "Löschen" können Sie auch die Wellenform ganz löschen. Solange das der Fall ist, hören Sie natürlich auch keine Töne. Denn wo nichts ist, kann man auch nichts hören.

Gefällt Ihnen eine Wellenform, können Sie sie mit einem Klick ins "Save"-Feld abspeichern. Geben Sie den Dateinamen an, das war schon alles. Und ein Klick in "Load" ermöglicht das Einlesen von gespeicherten Werten. Die so erzeugten Dateien können Sie in eigenen Programmen einlesen und für eigene Wellenformen verwenden. Die Einleseroutine muß so aussehen:

```
OPEN (Feldname) FOR INPUT AS 1
  FOR x=0 TO 256
    Wellenfeld(x)=127-ASC(INPUT$(1,1))
  NEXT x
CLOSE 1
```

Das Wellenfeld weisen Sie mit WAVE den gewünschten Stimmen zu. (Sie wissen ja noch, Sie können auch verschiedene Wellenformen für jede einzelnen Stimme angeben.) Jeder SOUND-Befehl auf der jeweiligen Stimme benutzt die eingelesene Welle.

Bleibt uns nur noch, Ihnen auch damit viel Spaß und viel Erfolg zu wünschen. Wie gesagt, mehr Musik steckt in AmigaBASIC leider nicht drin. Im Amiga natürlich schon, davon kann sich jeder überzeugen, der das eine oder andere professionelle Musikprogramm mal gehört hat. Damit ist das Sound-Kapitel auch zu Ende.

Im vierten Teil geht es nun noch um ein ganz anderes Thema. Nämlich darum, wie Sie Ihre BASIC-Programme durch einen *Compiler* schneller machen können.

IV. Der Turbo-Amiga

8. Sprachhistorie - Von Maschinensprache, Compilern und Interpretern

Eigentlich spricht Ihr Amiga überhaupt kein BASIC. Und er versteht es auch nicht. "Das ist ja die Höhe!" hören wir jetzt einige Leser aufschreien, "Da bringen uns die beiden auf über 500 Seiten AmigaBASIC bei, und dann behaupten sie, der Amiga würde überhaupt kein BASIC verstehen." Bevor solche Stimmen in eine allgemeine Protestbewegung umschlagen, wollen wir doch besser ganz schnell Licht in die Sache bringen:

Wie Sie in diesem Buch schon gelesen haben, ist für die gesamte Rechenarbeit im Amiga der 68000 zuständig. Der 68000 ist der Hauptprozessor des Amiga. Bei seiner Arbeit wird er von den Co-Prozessoren Agnus, Denise und Paula unterstützt, denen wir die überragenden Grafik- und Tonmöglichkeiten unseres Computers verdanken. Aber fürs Rechnen ist wie gesagt allein der 68000 zuständig. Und was tut ein Computer letzten Endes schon anderes als rechnen?

Jedes Programm ist eine Ansammlung von Rechen- und Steuerbefehlen. Und die einzige Programmiersprache, die unser 68000 versteht, ist *Maschinensprache*. Dieser Begriff tauchte schon an verschiedenen Stellen in diesem Buch auf und hatte immer ein wenig den Hauch des Komplizierten und Unerlernbaren um sich.

Der Grund dafür ist, daß Maschinensprache für Menschen sehr schwer zu lernen ist. Maschinensprache eben. Sie besteht streng genommen nämlich nur aus Nullen und Einsen. Eben den beiden Signalen, die ein Computer unterscheiden kann. Im Zwischenspiel 4 haben wir Ihnen gezeigt, mit welchen Tricks man aus solchen Nullen und Einsen schließlich doch größere Zahlen oder auch Buchstaben und Zeichen bilden kann.

Auch jeder Programmbefehl, den der 68000 verarbeiten soll, muß in Form von Nullen und Einsen vorliegen. Da Binärzahlen für Menschen aber extrem unübersichtlich und ungeheuer schwer zu merken sind, haben sich die Programmierer, die den 68000 direkt in Maschinensprache programmieren, leichter

merkbare Befehlskürzel einfallen lassen. Diese Befehlskürzel heißen "Mnemonics". Wir wissen, daß dieses Wort ziemlich schwer auszusprechen ist. Andererseits waren die Mnemonic-Sprachschöpfer wohl Schlimmeres gewohnt, denn die erwähnten Befehlskürzel sind auch nicht eben einfach auszusprechen. Um nur einige wenige Beispiele zu nennen: JSR, RTS, BSR, CMP.

Die meisten Mnemonics sind Abkürzungen für die englische Funktionsbeschreibung eines Befehls. JSR steht z.B. für "Jump SubRoutine" - "Unterprogramm aufrufen". Oder CMP für "CoMPare" - "Vergleichen". Wer mit solchen Kürzeln programmiert, arbeitet in der Programmiersprache *Assembler.* Das ist die eigentliche Programmierung in Maschinensprache.

Was diese Programmierung so schwierig macht, sind zwei Dinge: Erstens muß jede Programmfunktion in ungeheuer kleine Einheiten zerlegt werden. Das macht große Maschinenprogramme sehr fehleranfällig und erfordert eine umfassende Planung beim Programmieren. Und zweitens muß der Maschinensprache-Programmierer genau über jede Eigenschaft des Prozessors und der Co-Prozessoren Bescheid wissen. In Maschinensprache gibt es z.B. keinen Befehl, um eine Bildschirmfarbe zu verändern. Statt dessen muß unser Programmierer wissen, in welche Adresse welches Co-Prozessors er welchen Wert schreiben muß, damit die Hardware dann ihrerseits die Farbe verändert.

Einfach ist das alles nicht. Trotzdem muß irgendwann jeder Computer einmal in Maschinensprache programmiert werden. Und zwar dann, wenn sein *Betriebssystem* geschrieben wird. Nur ist das eben ein Job für absolute Profis, sogenannte Systemprogrammierer. Und selbst denen ist das Hantieren mit JSR, RTS & Co. nicht immer sehr angenehm. Ein fehlerfreies Betriebssystem gibt es wohl bis heute noch nicht...

Aus diesem Grund sind viele Maschinensprache-Programmierer schon bald auf die Idee gekommen, Programme zu schreiben, die ihnen diese unangenehme Arbeit abnehmen. Programme, die leichter merkbare Befehle in Maschinensprache übersetzen. Der Programmierer formuliert seine Wünsche an den Computer dann

etwas verständlicher und einfacher (z.B. Color = White) und das Übersetzungsprogramm erzeugt daraus automatisch die notwendigen Folgen von Maschinensprache-Befehlen.

Ein solches Übersetzungsprogramm nennt man Compiler. Compiler gibt es für viele verschiedene Programmiersprachen. Auf dem Amiga ist zum Beispiel die Sprache C sehr verbreitet und beliebt. Und zwar deshalb, weil sie noch relativ "maschinennah" arbeitet, also gute Kontrollmöglichkeiten über die Hardware bietet, und trotzdem flexibler und deutlich einfacher ist als Maschinensprache. Der Systemprogrammierer kann nun seine Programme in der Sprache C schreiben, der Compiler übersetzt das Programm dann und am Ende steht ein Programm in Maschinensprache.

Nun wird es Sie wohl nicht mehr überraschen, wenn wir Ihnen erzählen, daß große Teile des Amiga-Betriebssystems in C geschrieben wurden. Ein C-Compiler hat dann diese C-Befehle in für den 68000-Prozessor verständliche Maschinensprache übersetzt. Aus diesem Grund ist C bei Amiga-Programmierern übrigens so beliebt: Diese Sprache ist quasi die "Muttersprache" des Amiga und seines Betriebssystems. Mit ihr kann man fast alles auf dem Amiga realisieren.

Programmierer, die Anwendungsprogramme wie Deluxe Paint oder PageSetter schreiben, arbeiten daher meistens in C. Nur wenige von ihnen programmieren noch direkt in Maschinensprache.

Doch auch C ist noch vergleichsweise kompliziert und schwer zu lernen. Für Hobby-Programmierer und Einsteiger in die Welt des Programmierens wurden deshalb andere Programmiersprachen entwickelt. Wie eben zum Beispiel AmigaBASIC. AmigaBASIC ist sehr komfortabel, aber nicht mehr sehr maschinennah. Wir haben das im Verlauf des Buches zum Beispiel daran gemerkt, daß Funktionen, die nicht von vorneherein vorgesehen sind (etwa das Auslesen der Farbwerte oder das Laden von IFF-Bildern), nur unter großem Aufwand realisiert werden können.

Das Grundprinzip ist bei BASIC nicht anders als bei C: Die einzelnen Befehle müssen für den 68000-Prozessor in Maschinensprache-Befehle übersetzt werden. Jedoch sind die Befehle von BASIC wesentlich leistungsfähiger und komfortabler als die Befehle von C. Um den Preis, daß BASIC eben nicht mehr so nah an der Hardware ist.

Aber zwischen AmigaBASIC und C gibt es noch einen anderen wichtigen Unterschied. Wenn Sie einen AmigaBASIC-Befehl im BASIC-Window eingeben, wird der Befehl sofort ausgeführt. Und wenn Sie ein Programm ins LIST-Window eingeben und starten, fängt der Amiga sofort an, das Programm abzuarbeiten. Wenn er dabei einen Fehler entdeckt, hält er das Programm an und bringt eine Fehlermeldung. Das alles funktioniert nur, weil BASIC eine sogenannte "Interpreter"-Sprache ist. Das englische Wort "Interpreter" wird ungefähr "Intörrpräterr" ausgesprochen und bedeutet übersetzt "Dolmetscher". Und genauso können Sie sich AmigaBASIC auch vorstellen: Wie ein Dolmetscher einen Satz hört und übersetzt, findet AmigaBASIC eine Programmzeile vor und übersetzt sie in Maschinensprache. Dieser Vorgang findet während der Abarbeitung des Programms statt.

Im Gegensatz dazu arbeitet ein Compiler eher wie der Angestellte eines Übersetzungsbüros: Er übersetzt den ganzen Text auf einmal und gibt ihn dann weiter an den Adressaten. Sollten beim Übersetzen irgendwelche Unklarheiten oder Probleme auftreten, wird der Urheber des Textes vorher gefragt. Beim Compiler läuft's nun ganz genauso: Er übersetzt das komplette Programm und erzeugt dann ein eigenes Programm in Maschinensprache (die Übersetzung). Treten beim Compilieren Fehler oder Unklarheiten auf, bricht der Compiler ab und meldet dem Programmierer, daß in dem zu übersetzenden Programm (engl.: Source-Code, deutsch: Quell-Programm) etwas unklar ist.

Der Nachteil bei Interpreter-Sprachen ist, daß die Übersetzung während des Programmablaufs erfolgt. Befehl für Befehl, Zeile für Zeile. Die Zeit, die AmigaBASIC zum Übersetzen benötigt,

verlangsamt logischerweise Ihr Programm. Programme, die von einem Interpreter abgearbeitet werden, sind deshalb immer langsamer als compilierte Programme.

Vielleicht fragen Sie sich jetzt, welchen Vorteil es dann hat, daß AmigaBASIC als Interpreter arbeitet. Ganz einfach: Interpretersprachen sind einfacher zu handhaben: Sie können Ihr BASIC-Programm eintippen, ausprobieren und abwarten, wo ein Fehler auftritt. Den Fehler verbessern Sie dann und starten das Programm von Neuem. Diese "Trial and Error"-Methode - Ausprobieren und aus den Fehlern lernen - ist für BASIC besonders typisch.

Bei einer Compiler-Sprache wird das Ganze wesentlich aufwendiger: Vielleicht merkt der Compiler einen Fehler erst nach mehreren Minuten Übersetzungszeit. Dann muß der Programmierer den Compiler verlassen, sein Quellprogramm laden, nach dem Fehler suchen, ihn verbessern, die verbesserte Version abspeichern, den Compiler neu starten und hoffen, daß diesmal kein Fehler mehr auftritt. So gesehen ist AmigaBASIC als Interpretersprache also wesentlich angenehmer. Was bleibt, ist der Nachteil der Geschwindigkeit. AmigaBASIC gehört zwar zu den Schnellsten seiner Art, aber Sie haben sicher schon an der einen oder anderen Stelle gemerkt, daß es manchmal ganz schön auf sich warten läßt.

8.1 Machen Sie Ihren BASIC-Programmen Beine - der AC/BASIC-Compiler

Die Nachteile, die AmigaBASIC als Interpretersprache hat, brachten die amerikanische Softwarefirma Absoft auf eine Idee: Sie hat für AmigaBASIC einen Compiler geschrieben. Dieser Compiler hat den Namen "AC/BASIC" und kostet etwa zwischen DM 200,- und DM 300,-. Wenn Sie sich dieses Programm kaufen, haben Sie die Möglichkeit, Ihre AmigaBASIC-Programme zu compilieren. Das heißt, der Compiler übersetzt ein ganz normales AmigaBASIC-Programm in ein Maschinensprache-Pro-

gramm. Unter bestimmten Voraussetzungen kann man damit einen beachtlichen Geschwindigkeitsgewinn erzielen - eben seinen BASIC-Programmen Beine machen.

In diesem vierten Teil unseres Buchs wollen wir Ihnen nun die Möglichkeiten des AC/BASIC-Compilers vorstellen. Wir wollen Ihnen zeigen, wann sich der Kauf und Einsatz dieses Programms lohnt, und wann nicht. Und wir wollen Ihnen einige Tips geben, die Sie beachten sollten, wenn Sie die Beispielprogramme aus diesem Buch compilieren wollen. Das soll Ihnen später helfen, wenn Sie selbstgeschriebene Programme compilieren und dabei Schwierigkeiten und Probleme auftreten.

Um beurteilen zu können, wann ein Compiler für Ihre Programme einen deutlichen Geschwindigkeitsgewinn bringt, sollten Sie noch einiges über die Funktionsweise von AmigaBASIC und die Arbeit des Compilers erfahren.

Sie wissen ja bereits, daß AmigaBASIC selbst ein Programm ist. Zumindest haben wir das schon mal ganz am Anfang unseres Buchs erwähnt. Wenn Sie ein BASIC-Programm eingegeben haben und es starten, arbeitet das Programm "AmigaBASIC" (also der Interpreter) Ihr Listing Befehl für Befehl ab. Für jeden Befehl gibt es dabei einen Programmteil in "AmigaBASIC". Das Ganze arbeitet also ähnlich wie etwa unser Malprogramm: Wenn Sie dort eine bestimmte Funktion anwählten, etwa "Kreise", dann kümmerte sich der Programmteil "Kreis:" darum.

Ähnlich gibt es in "AmigaBASIC" Programmteile, die in Aktion treten, wenn in Ihrem Listing z.B. ein CIRCLE-Befehl, ein SCREEN-Befehl oder auch eine FOR...NEXT-Schleife vorgefunden werden. Für jeden BASIC-Befehl ist ein Teil des Interpreter-Programms zuständig. Und wie in unserem Malprogramm treten die einzelnen Programmteile wenn nötig untereinander in Verbindung. Dieser Vorgang läuft dann solange ab, bis Ihr BASIC-Programm abgearbeitet ist.

Allerdings muß sich AmigaBASIC erfreulicherweise nicht um alles selbst kümmern. Viele Funktionen nimmt ihm nämlich das Betriebssystem ab. Wenn Sie beispielsweise ein neues Window öffnen, ruft AmigaBASIC einfach die dafür zuständige Betriebssystemroutine auf. Und das ist ja, wie Sie im letzten Kapitel erfahren haben, eine Routine in Maschinensprache, die durch einen C-Compiler erzeugt wurde. Da der Amiga eine grafische Benutzeroberfläche hat und vom ganzen System her eine Grafik-Maschine ist, finden sich im Betriebssystem besonders viele Routinen, die mit Grafik zu tun haben. Zum Beispiel zur Verwaltung von Windows und Screens, zum Zeichnen von Linien, Rechtecken, Polygonen und zum Ausmalen solcher Figuren. In all diesen Fällen ruft AmigaBASIC einfach Routinen aus dem Betriebssystem auf.

Andere Dinge, wie das Verwalten von FOR...NEXT- oder WHILE...WEND-Schleifen oder das Berechnen mathematischer Ausdrücke kann AmigaBASIC nicht ans Betriebssystem deligieren. Hier muß es dann selbst in Aktion treten.
Wie arbeitet nun der AC/BASIC-Compiler? Im Prinzip nicht viel anders als AmigaBASIC. Er analysiert beim Compilieren Ihr BASIC-Programm und erzeugt ein Maschinensprache-Programm, das genau dieselben Funktionen erfüllt. Trifft er auf einen Befehl, der vom Betriebssystem erledigt werden kann, fügt er in das Maschinensprache-Programm einen entsprechenden Betriebssystem-Aufruf ein. Findet er einen oder mehrere Befehle, die zur Programmsteuerung, Abarbeitung von Schleifen oder Berechnung mathematischer Ausdrücke dienen, verwendet er im "Object-Code" (so nennt man das Ergebnis des Compilierens) einen oder mehrere Maschinensprache-Befehle, die dasselbe bewirken.

Der Compiler versucht dabei zu vermeiden, daß sich im Object-Code immer wieder dieselben Routinen wiederholen. Stellen Sie sich vor, in Ihrem Quell-Programm kommt an zwanzig verschiedenen Stellen eine SIN-Berechnung vor, und der Compiler würde daraufhin im Object-Code an zwanzig verschiedenen Stellen ein Maschinenprogramm zum Ausrechnen von Sinus-Werten einfügen. Das wäre ja nun wirklich Verschwendung von

Speicherplatz. Aus diesem Grund benutzt auch der Compiler Unterprogramme. Er hat dazu eine Bibliothek fertiger Funktionen, die sogenannte Runtime-Library.

Langsam nähern wir uns der Frage: Wann kann der Compiler eigentlich einen deutlichen Geschwindigkeitsvorteil erreichen? Also, beim Aufrufen der Betriebssystem-Funktionen sicher nicht. Die sind nämlich immer gleich schnell, egal ob sie von AmigaBASIC oder einem Object-Programm in Maschinensprache aktiviert werden. Wenn wir unterstellen, daß die Berechnungen in Maschinensprache schneller vonstatten gehen als AmigaBASIC das hinbekommt, dann ist da schon eher eine Beschleunigung denkbar. Und bei Schleifen, Programmverzweigungen etc., die in Maschinensprache relativ einfach nachgebildet werden können, kann der Compiler auch noch was herausholen.

Das alles bedeutet also: Wenn Sie ein Programm durch den Compiler beschleunigen wollen, dann bringt das am meisten, wenn das Programm sehr rechenintensiv ist und aus vielen Schleifen und Sprungbefehlen besteht. Programme, die viele Grafikbefehle benutzen, werden Sie hingegen auch durch den Compiler kaum schneller machen können. Auch Programme, die in erster Linie Daten von Diskette lesen oder Daten auf Diskette schreiben, werden kaum schneller werden. Der Grund ist hier, daß die meiste Zeit durch das Lesen und Schreiben im Diskettenlaufwerk verlorengeht. Und daran kann auch der Compiler nichts ändern.

Wir haben auf der beiliegenden Diskette in der "Compiler"-Schublade die compilierten Versionen einiger Beispielprogramme aus unserem Buch abgelegt. So können sich auch diejenigen, die AC/BASIC nicht besitzen, ein Bild von der möglichen Beschleunigung machen.

Für alle Besitzer von AC/BASIC und diejenigen, die sich für diesen BASIC-Compiler interessieren, zeigen wir in den nächsten Kapiteln, wie dieser Compiler grundsätzlich bedient wird (denn leider gibt es im Moment nur ein englisches Handbuch dazu und laut Auskunft des Herstellers ist kein deutsches Manual geplant),

wir wollen Ihnen Tips geben, was bei der Arbeit damit zu beachten ist, und was Sie wissen müssen, wenn Sie bestimmte Beispielprogramme aus unserem Buch compilieren wollen.

8.2 Eins, zwei, drei im Sauseschritt - Vorbereitungen und die Bedienung von AC/BASIC

Die jetzt folgenden Anweisungen brauchen natürlich nur diejenigen unter Ihnen auch ausführen, die den Compiler schon besitzen. Die anderen sollten die Anweisungen einfach überspringen.

Fangen wir einfach mal mit einem Beispiel an. Angenommen, Sie wollen unser Videotitel-Programm compilieren. Was müssen Sie dann tun?

Die erste Voraussetzung, die AC/BASIC an ein BASIC-Programm stellt, ist das Format, in dem das Programm vorliegt. AC/BASIC verarbeitet nämlich nur ASCII-Dateien. Das bedeutet, daß Sie die Version für den Compiler mit SAVE "(Programmname)",A abspeichern müssen.

- Bitte laden Sie "Videotitel.IFF". Sie finden diese Programmversion z.B. in der "Daten"-Schublade auf der beiliegenden Diskette.

- Speichern Sie das Programm nun bitte als ASCII-Datei ab. Um die Zeiten für Schreiben und Lesen von Diskette beim Compilieren zu verkürzen, könnten Sie die ASCII-Version auf der RAM-Disk abspeichern. Geben Sie also bitte im BASIC-Window ein:

```
save "ram:Videotitel.IFF",a
```

- Nun können Sie AmigaBASIC wieder verlassen.

- Starten Sie dann bitte den AC/BASIC-Compiler.

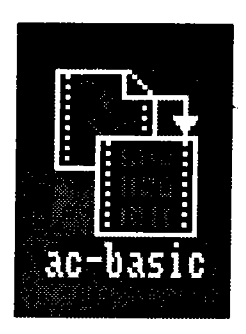


Bild 21: Das Icon von AC/BASIC

Vorher müssen Sie den Compiler, so wie im AC/BASIC-Handbuch erklärt, auf einer Arbeitsdiskette, oder - falls vorhanden - auf Ihrer Festplatte installieren. Sollte es später während der Anwendung Probleme geben und in AC/BASIC die Fehlermeldung "Overlay: 01.bc not found" oder eine ähnliche Meldung erscheinen, dann haben Sie vermutlich vergessen, den Inhalt des L-Verzeichnisses ins L-Verzeichnis Ihrer Workbench-Diskette zu kopieren.

Bei dieser Gelegenheit ein Wort zum Handbuch von AC/BASIC. Wenn Sie AC/BASIC besitzen, sollten Sie dieses Handbuch auf jeden Fall durchlesen. Es ist sehr gut geschrieben und enthält eine Menge nützlicher Hinweise. Das einzige Problem ist wie gesagt, daß dieses Handbuch bisher nur auf Englisch vorliegt. Wenn Sie AC/BASIC gestartet haben, erscheint zunächst ein Window, in dem folgendes zu lesen ist:

```
Absoft AC/BASIC Compiler

Version 1.2  April 10, 1987

Copyright (c) Absoft Corporation 1985, 1986, 1987
```

Die Versionsnummer entspricht zumindest der Version, die uns vorlag. Vielleicht ist sie bei Ihnen schon höher und ein späteres Datum steht daneben. Aber so etwas kennen Sie ja schon.

Wenn Sie die rechte Maustaste drücken, werden Sie entdecken, daß dieses Window ein einziges Pulldown besitzt. Es hat den Namen "Project".

- Bitte wählen Sie "Open" aus dem "Project"-Pulldown.

- Auf dem Bildschirm erscheint nun ein Requester, in dem Sie den Namen des Programms eingeben können, das Sie compilieren wollen. Geben Sie bitte ein:

```
ram:Videotitel.iff <RETURN>
```

Der Druck auf die <RETURN>-Taste erübrigt das Klicken ins OK-Feld. Wenn Sie sich beim Dateinamen nicht vertippt haben, erscheint nun das Haupt-Window von AC/BASIC.

Absoft AC/BASIC Compiler

SOURCE FILE:BASICDisk:Compiler/Malp WORK AREA: 32K

Compile

CANCEL

Compile Batch

Compiler Options

A Use Long Addressing
C Enable Run-time Tests
D Compile for Decimal Math
E Generate Errors List
I List Include Statements
L Generate Full List
N Process Run-time Events
R Link Run-time
S Generate Symbol File
T Temporaries on RamDisk
U Default Arrays to STATIC

SAVE

CLEAR

Bild 22: **Das Haupt-Window von AC/BASIC**

In diesem Window finden Sie alle Bedienungselemente von AC/BASIC. Die Bedienung des Programms ist übrigens besonders einfach. Sie brauchen also keine detaillierten Kenntnisse, um das Programm dazu zu bringen, etwas zu compilieren.

Unterhalb der Schrift "Compiler Options" sehen Sie verschiedene Felder. Sie alle entsprechen verschiedenen Optionen von AC/BASIC. Drei Felder sind mit einem orangen Kästchen gekennzeichnet: "C Enable Run-time Tests", "N Process Runtime Events" und "T Temporaries on RamDisk". Um die einzelnen Optionen und ihre Bedeutung werden wir uns etwas später kümmern.

- Bitte klicken Sie noch zusätzlich ins Kästchen "R Link Run-time".

Daß eine Option aktiviert ist, erkennen Sie daran, daß AC/BASIC das Kästchen vor der Option orange ausmalt.

- Bitte klicken Sie nun ins Feld "Compile".

Daraufhin verschwindet das Haupt-Window, und AC/BASIC beginnt, das Videotitel-Programm zu compilieren. Das wäre es auch schon gewesen, wenn das Wörtchen wenn nicht wär... Denn praktisch kaum ein Basic-Programm läßt sich so einfach compilieren. Kleinere Änderungen muß man häufig durchführen. Und damit man auch weiß welche, gibt der Compiler eine Reihe von Informationen während und nach dem eigentlichen compilieren. In einem leeren Window erscheinen deshalb verschiedene Ausgaben des Programms.

Zunächst ist da zu lesen:

```
0: ram:Videotitel.iff
```

Das ist der Name des Quell-Programms, das AC/BASIC gerade einliest. Darunter zählt das Programm ziemlich schnell von 0 bis 443. Diese Zahlen entsprechen den eingelesenen Zeilen.

Kurz darauf können Sie lesen:

```
1: Symbol table complete - 1 error detected
    Memory usage:
        Labels      4860 bytes
        Symbols     3262 bytes
        Total       59308 bytes
        Excess      9340 bytes
        Source      443 lines
2: Bypassing
3: Error report:

    CLS : PRINT "Bitte Dateinamen eingeben:"

*****  error in line 00368 [ CLS :]:  56 - Statement not in a subprogram
block

4: Bypassing
```

Aha. Na, da kamen ja eine ganze Menge Informationen. Wir wollen mal der Reihe nach sehen, was sie zu bedeuten haben. Zunächst mal die Zahlen 0:, 1:, 2:, 3: und 4:. Sie entsprechen den einzelnen Stationen, die unser Programm beim Compilieren durchläuft. AC/BASIC ist ein sogenannter 2-Pass-Compiler. Das bedeutet, daß er das Quell-Programm zum Compilieren zweimal durchläuft. Der erste Pass (deutsch: Durchlauf) fand bei der Nummer 1: statt. Zum zweiten Pass ist es allerdings gar nicht mehr gekommen. AC/BASIC hat nämlich einen Fehler entdeckt. Deshalb steht auch am Ende der Zeile "1 error detected".

Pass 1 dient AC/BASIC dazu, das Programm nach verschiedenen Kriterien zu untersuchen. Stellt der Compiler dabei einen Error fest, wird nicht compiliert. Den Fehler erklären wir Ihnen etwas später. Jetzt ist nämlich erstmal die "Memory Usage" dran. Der Speicherverbrauch also. AC/BASIC gibt uns hier einige Informationen, die alle mit dem Speicherverbrauch des aktuellen Quellprogramms zu tun haben.

Labels

Zum Compilieren muß AC/BASIC sämtliche Labels kennen, die im Quellprogramm vorkommen. Dazu legt der Compiler eine Label-Tabelle an. Die Größe dieser Tabelle in Bytes wird hinter "Labels" angezeigt.

Symbols

Auch die Variablen, die im Programm vorkommen, müssen von AC/BASIC in einer eigenen Tabelle verwaltet werden. Die Angabe hinter "Symbols" teilt uns die Größe dieser Tabelle in Bytes mit.

Total

Hier sagt uns AC/BASIC wieviel Speicher es selbst zusammen mit seinem Arbeitsspeicher benötigt. Vom gesamten Speicher des Amiga belegt AC/BASIC inklusive seiner Daten nun also knapp 60 KByte. Wenn man die Leistungsfähigkeit des Programms bedenkt, ist das ziemlich wenig. Finden Sie nicht?

Excess

Hier steht, wieviele Bytes vom Arbeitsspeicher noch nicht benötigt wurden. Die Größe des Arbeitsspeichers läßt sich übrigens mit einem Schieberegler im Haupt-Window einstellen. Wenn es mal beim Compilieren Speicherprobleme gibt, dann können Sie diesen Wert einfach vergrößern.

Source

Hier stehen zur Abwechslung mal keine Bytes, sondern "lines". Diese Zahl gibt an, aus wie vielen Zeilen Ihr Quell-Programm besteht. Da Sie diesen Wert bei AmigaBASIC sonst nie erfahren, kann man da selbst manchmal ganz schön überrascht werden. Oder hätten Sie richtig geschätzt, daß unser Videotitel-Programm aus 443 Zeilen besteht?

Im Schritt "2:" findet normalerweise Pass 2 statt. Da der aber aufgrund des gefundenen Fehlers ausfallen muß, steht hier "Bypassing". Das bedeutet soviel wie "Umleitung" und zeigt an, daß es ohne zweiten Durchlauf weitergeht.

In 3: erfahren wir nun endlich, wo das Problem liegt, das AC/BASIC beim Compilieren gefunden hat. Der "Error Report", der "Fehlerbericht" also, ergibt in unserem Fall folgendes:

```
CLS : PRINT "Bitte Dateinamen eingeben:"
```

Zuerst zeigt AC/BASIC den Inhalt der Zeile, in der der Fehler auftrat. Dann erfahren wir, daß dieser Fehler in Zeile 368 auftrat, und daß der auslösende Befehl das CLS war. Nun stellt sich natürlich die Frage, was denn wohl nicht stimmen mag. In der abgedruckten Zeile läßt sich ja beim besten Willen kein Fehler entdecken.

Mal sehen, was die Fehlermeldung besagt. Da steht "Statement not in a subprogram block". Der Befehle befindet sich also nicht innerhalb eines SUB...END SUB-Blocks. Na und? Das muß er ja wohl auch nicht, oder? Nun, um Sie nicht lange auf die Folter zu spannen: Für AC/BASIC muß er schon. Eine weitere Anforderung, die der Compiler an seine Quell-Programme stellt, ist nämlich, daß sämtlich SUB-Routinen am Programmende stehen müssen. Bei AmigaBASIC ist das ja nicht unbedingt so - da können SUB...END SUB-Blocks nach Belieben auch irgendwo mitten im Programm stehen.

Aus organisatorischen Gründen muß AC/BASIC da etwas pingeliger sein. Uns bleibt also nichts übrig, als den SUB...END SUB-Block in unserem BASIC-Programm ans Programmende zu bringen. Wir wollen das in diesem Fall selbst erledigen. Auf der "AC/BASIC"-Programmdiskette finden Sie aber auch ein BASIC-Programm namens "sortsubs", das Ihnen diese Arbeit abnimmt. Das dauert dann zwar relativ lang, kann aber bei sehr langen und komplexen Programmen helfen, Ihre Nerven nicht über Gebühr zu strapazieren.

Machen wir uns gleich an die nötigen Umbau-Arbeiten. Sie brauchen AC/BASIC dazu nicht zu verlassen. Verkleinern Sie das Window einfach ein wenig und schieben Sie es in eine Ecke des Bildschirms, wo es Sie nicht stört.

- Starten Sie dann bitte AmigaBASIC.

Wenn Sie das Window der "BASICDisk" vorhin geschlossen haben, öffnen Sie es bitte wieder und starten Sie AmigaBASIC durch einen Doppelklick. Sollten Sie bisher alles so gemacht haben, wie von uns vorgeschlagen, liegt die ASCII-Version des Videotitel-Programms zur Zeit auf der RAM-Disk.

- Laden Sie bitte dieses Programm, indem Sie im BASIC-Window eingeben:

```
load "ram:Videotitel.iff"
```

Sobald das Programm geladen ist und im LIST-Window erscheint, geben Sie bitte im BASIC-Window ein:

```
list Schleife6:
```

Damit gelangen Sie in unmittelbare Nähe des SUB-Programms "Laden". Das ist der Übeltäter, den wir ans Programmende befördern müssen.

- Bitte markieren Sie den Programmteil ab der Zeile

```
SUB Laden STATIC
```

mit der Maus. Halten Sie die Maus solange gedrückt und scrollen Sie nach unten durch das Listing, bis Sie bei der Zeile

```
END SUB
```

im Programteil "EndeLaden:" angekommen sind.

- Schneiden Sie den markierten Programmteil dann bitte mit der Option "Cut" aus dem "Edit"-Menü aus.

Es wird einen Augenblick dauern, bis der Amiga diesen Riesenbrocken geschluckt hat. Aber dann verschwindet das gesamte SUB-Programm aus dem Listing. Es befindet sich nun im Clipboard.

- Bitte springen Sie nun mit <ALT> <Cursor nach unten> ans Programmende. Dort fügen Sie dann bitte das ausgeschnittene SUB-Programm mit der Option "Paste" aus dem "Edit"-Pulldown wieder ein.

Das war schon alles. Das SUB-Programm steht nun ordnungsgemäß am Ende des Listings. Wenn Sie in Zukunft Programme von vorneherein mit der Absicht schreiben, sie zu compilieren, können Sie die SUB-Programme ja immer gleich ans Ende schreiben.

- Um die korrigierte Version abzuspeichern, geben Sie bitte im BASIC-Window ein:

```
save "ram:Videotitel.IFF",a
```

- Verlassen Sie dann wieder AmigaBASIC.

- Suchen Sie nun auf dem Bildschirm das AC/BASIC-Window und ziehen Sie es wieder auf volle Größe.

- Wählen Sie dann erneut "Open" aus dem "Project"-Menü. Im daraufhin erscheinenen Lade-Requester sehen Sie noch immer den alten Programmnamen "ram:Videotitel.iff" stehen. Klicken Sie also einfach ins "OK"-Feld, um die neue Version zu laden.

Nun erscheint wieder das Haupt-Window von AC/BASIC. Die Optionen sind alle noch so eingestellt wie vorhin. In der rechten oberen Bildschirmecke sehen Sie nun auch den Schieberegler für den Arbeitsspeicher, von dem wir vorhin gesprochen haben. Für das aktuelle Programm reichen die 32 KByte, die als Minimum eingestellt sind, voll aus. Wenn Sie jedoch an Speicherplatz nicht arm sind (also so um 1 MByte RAM oder mehr besitzen), dann können Sie ja mal testhalber einen anderen Wert einstellen. Bis

zu 224 KByte reicht der Schieberegler. Das ist wichtig für Programme, die sehr viele Variablen oder Labels beinhalten, sehr groß sind, oder aus vielen Unterprogramm bestehen.
Aber wie gesagt: Für das Videotitel-Programmen sind die standardmäßigen 32 KByte bereits mehr als genug.

- Klicken Sie zum Compilieren ins Feld "Compile".

Wieder erscheinen Ausgaben im ansonsten leeren Window von AC/BASIC.

Diesmal findet der Pass 1 jedoch keinen Fehler. Nach der Meldung "Symbol table complete" folgt die bereits bekannte Auskunft über die Speicherbelegung. Und dann geht's folgendermaßen weiter:

```
2: Object file complete
3: Program file complete: 24628 Bytes
   Stack Size: 4480 Bytes
   Elapsed time: 0:35 = 761 lines/minute
```

Pass 2 meldet also, daß die erzeugte Objekt-Datei vollständig ist. Und bei 3: erfahren Sie noch einiges zum abgespeicherten Ergebnis des Compilierens. Die Zahl hinter "Program file complete:" gibt die Größe des erzeugten Maschinenprogramms an. Unser Object-Programm wird jedoch nachher deutlich größer sein, weil wir die Runtime-Library dazugebunden haben. Doch dazu später mehr. Die "Stack Size" gibt an, wie viele Bytes Stack-Speicher das erzeugte Programm benötigt.

Hinweis: Wenn Sie Ihre Programme von der Workbench-Oberfläche aus starten, brauchen Sie sich um die Stack-Größe nicht zu kümmern. Nur CLI-Anwender, die ihre Programme direkt aus dem CLI aufrufen, müssen hier aufpassen: Wenn der Stack-Bedarf eines Programms größer ist, als der vom CLI bereitgestellte Stack-Speicher, läuft das Programm nicht. Mit ein wenig Pech kann das bis zu einer "Guru

Meditation" führen. Wenn Ihnen der Compiler also einen Stack-Bedarf meldet, der die standardmäßigen 4000 Bytes des CLI übersteigt, dann vergrößern Sie bitte diesen Speicherbereich mit dem CLI-Befehl STACK.

Zu guter Letzt schließlich teilt Ihnen AC/BASIC, nicht ganz ohne Stolz, mit, wie lange es zum Compilieren gebraucht hat. In unserem Fall waren das 35 Sekunden. Das ergibt einen Schnitt von 761 Zeilen pro Minute. Das ist schon ganz schön fix. Wir kennen jedenfalls keinen Übersetzer, der auf so einen Durchschnittswert kommt.

Tja. Damit haben Sie Ihr erstes BASIC-Programm compiliert. War doch ganz einfach, oder? Sie können AC/BASIC jetzt erstmal durch Anwählen der "Quit"-Option im "Project"-Pulldown verlassen. Schauen wir uns jetzt mal das Ergebnis des Compilierens an. Sie finden es auf der RAM-Disk unter dem Namen "Videotitel.IFF.run". Das Icon mit seinen Nullen und Einsen zeigt ja schon recht deutlich, daß Sie es jetzt mit einem Maschinenprogramm zu tun haben.

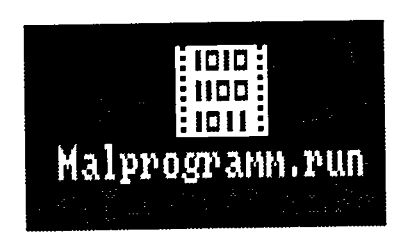


Bild 23: **Das Icon eines compilierten Programms**

Dieses Programm können Sie anklicken, wie Sie es von allen Amiga-Programm gewohnt sind. Da es völlig unabhängig von AmigaBASIC, AC/BASIC oder irgendwelchen anderen Programmen läuft, könnten Sie es auch auf jede beliebige andere Diskette kopieren. Aus dem BASIC-Programm wurde ein waschechtes, vollwertiges Amiga-Maschinenprogramm.

Bitte kopieren Sie das Programm zunächst von der RAM-Disk auf eine Arbeitsdiskette. Auch die Quell-Version "Videotitel.IFF" sollten Sie kopieren. Ungesicherte Programmversionen fristen nämlich auf der RAM-Disk ein ziemlich gefährliches Dasein. Sollte es aus irgendwelchen Gründen zum Systemabsturz kommen, war Ihre ganze Arbeit umsonst. Und Systemabstürze sind bei den compilierten Versionen leider gar nicht so selten.

Bevor Sie das Programm nun ausprobieren, haben wir noch einige Anmerkungen. Erstens: Bitte benutzen Sie den Programmpunkt "Hintergrundbild festlegen" noch nicht! Die gegenwärtige Version stürzt beim Einlesen eines IFF-Bilds ab. Warum das so ist, und was man dagegen tun kann, erfahren Sie im nächsten Kapitel.

Wir haben aber auch eine gute Nachricht: Wenn Sie ein Grafikobjekt einlesen, z.B. "BASICDisk:Videotitel/Star" und dieses Objekt dann auf dem Bildschirm positionieren, wird Ihnen auffallen, daß das Objekt nun gar nicht mehr flackert. Bei Programmen, die Sie mit AC/BASIC compiliert haben, fällt das Flackern der Bobs auf PAL-Amigas weg! Toll, nicht? Das liegt übrigens vermutlich daran, daß Absoft zur Objekt-Bewegung die entsprechenden Betriebssystem-Routinen verwendet, wohin Microsoft eigene zeitkritische Routinen für AmigaBASIC geschrieben hat.

Sie können ja nun mal einen Videotitel erzeugen. Beobachten Sie dabei, an welchen Stellen das Programm schneller ist als vorher. Und bitte halten Sie sich zunächst noch vom Einlesen von Hintergrundbildern fern.

Nur einen Hinweis sollten wir Ihnen noch geben: Nämlich wie Sie das compilierte Programm wieder verlassen können. Das ist allerdings nicht allzu schwierig. Wenn Sie die rechte Maustaste drücken, entdecken Sie im compilierten Programm ein "Project"-Pulldown. Dessen einziger Menüpunkt ist "Quit". Wählen Sie also "Quit", um das Programm zu beenden. Sie können statt dessen übrigens auch ins Schließsymbol des Windows klicken. Wenn das Programm gerade auf eine Tastatureingabe wartet, also der

Cursor hinter einem Fragezeichen steht, müssen Sie zusätzlich nach der Anwahl von "Quit" noch diese Eingabe beenden. Drücken Sie dazu einfach die <RETURN>-Taste.

8.3 Die Kammerjäger im Computer - Debugging

Das hat ja bisher alles ganz gut geklappt. Aber so völlig problemlos ist das Compilieren von BASIC-Programmen nun auch wieder nicht. Wir wollen Ihnen das zunächst beweisen.

Bitte speichern Sie alle ungesicherten Programme und Daten ab. Beenden Sie dann alle Programme, die Sie vielleicht im Rahmen von Multitasking benutzen. Und starten Sie dann die compilierte Version "Videotitel.IFF.run". Dieses Experiment klappt übrigens nur, wenn Sie den AC/BASIC-Compiler selbst besitzen, und das Beispiel aus dem letzten Kapitel selbst compiliert haben. In den Versionen auf der beiliegenden Diskette haben wir nämlich die Fehler soweit irgend möglich behoben.

Wenn das Programm läuft, erstellen Sie bitte einen kleinen Videotitel. Geben Sie sich dabei aber nicht zuviel Mühe, das Ergebnis wird nämlich nicht lange überleben. Wählen Sie dann mit Menüpunkt 6 die Option "Hintergrundbild festlegen" und geben Sie dort ein, daß hinter Ihrem Videotitel irgendein IFF-Bild gezeigt werden soll.

Ja, und dann starten Sie den Titel mit Option 5. Zuerst klappt alles wie gewohnt: Der Countdown zählt rückwärts von 10 nach 0. Sobald er dort angekommen ist, läuft das Diskettenlaufwerk an, von dem Ihr IFF-Bild geladen werden soll. Doch dann ist es nur noch eine Frage von Sekunden, bis die Meldung

```
Software error - task held
 Finish ALL disk activity
Select CANCEL to reset/debug

        OK     CANCEL
```

auf dem Bildschirm erscheint. Vielleicht kam es auch gar nicht mehr so weit und die berühmt-berüchtigte "Guru Meditation" erscheint gleich auf dem Bildschirm. Drücken Sie bitte die rechte Maustaste, um den "Software error" zu bestätigen und dann die linke Maustaste, um nach dem "Guru" Ihren Amiga neu zu booten.

Was Sie da eben erlebt haben, nennen die Software-Leute einen "Bug". Dieses englische Wort bedeutet auf deutsch "Käfer, Wanze". Gemeint ist aber ein Programmfehler. Stellt sich die Frage, was die kleinen Tierchen mit einem Programmabsturz zu tun haben sollen. Darüber gibt es eine sehr beliebte Geschichte aus den Anfängen der Computerei. Seinerzeit gab es nämlich noch keine Mikrochips. Die Computer der ersten Generation waren größtenteils aus Röhren aufgebaut und füllten leicht ganze Fabrikhallen. Daß die damals trotzdem noch deutlich weniger leisteten als heute zum Beispiel ein VC-20, haben Sie ja sicher auch schon gehört. Trotzdem wurden diese Maschinen schon fleißig eingesetzt, um z.B. mathematische Funktionen zu berechnen oder die Lieblingszahlen der Mathematiker wie e oder Pi auf mehrere hundert Stellen auszurechnen.

Als nun eines Tages eines dieser Programme mal wieder den Geist aufgab (und mit ihm ein Teil des Computers), entdeckten die Techniker einen kleinen Käfer, der es sich in einer der Schaltungen bequem gemacht hatte. Seitdem ist "Bug" die Standardbezeichnung für Softwarefehler. Und der Vorgang des Fehler-Behebens heißt konsequenterweise "Debugging" - "Entwanzen". Aus diesem Grund können sich Programmierer also mit Fug und Recht als Computer-Kammerjäger bezeichnen.

Widmen wir uns also dem Bug, der unser Programm so richtig satt zum Absturz gebracht hat. Der Absturz ist ja eigentlich sehr seltsam, denn in der BASIC-Version lief das Programm doch noch einwandfrei. Es muß also durch das Compilieren irgendetwas schiefgelaufen sein.

Mit diesen Überlegungen machten wir uns auf die Suche. Bei einigen anderen Experimenten, mit dem Malprogramm oder der IFF-Leseroutine stellten wir fest, daß die compilierten Versio-

nen immer dann abstürzen, wenn sie den BODY-Chunk einer IFF-Datei einlesen sollen. Die traurige Nachricht ist also, daß zumindest die gegenwärtig vorliegende Version von AC/BASIC (Version 1.2 vom 10. April 1987) noch einige gravierende Fehler beim Compilieren macht. Und diese Fehler können recht schnell zu einem Systemabsturz führen.

Bei unseren weiteren Experimenten haben wir den Fehler schließlich eingekreist: Die Programme sind immer dann abgestürzt, wenn die Funktion CVL eingesetzt wurde. Sie erinnern sich: Mit CVL ("Convert Long") wandeln wir in unserer IFF-Routine 4 eingelesene Bytes in eine 32-Bit-Zahl um, die wir dann mit POKEL in den Speicher bringen.

Uns bleibt also bei der gegenwärtigen Version von AC/BASIC nichts anderes übrig, als auf CVL zu verzichten. Das ist glücklicherweise nicht ganz so schlimm, weil der Befehl CVI (Umwandeln von 2 Bytes in einen 16-Bit-Wert) auch nach dem Compilieren ordnungsgemäß funktioniert.

Um die IFF-Leseroutine auch nach dem Compilieren funktionsfähig zu halten, müssen wir also eine Änderung vornehmen.

- Bitte laden Sie das Quellprogramm "Videotitel.iff" in AmigaBASIC.

- Geben Sie im BASIC-Window ein:

```
list BodyMap:
```

- Ändern Sie den Programmteil

```
FOR x1=0 TO 9
  POKEL Ebnadr(b)+4*x1+40*y1,CVL(INPUT$(4,1))
NEXT x1
```

so, daß er folgendermaßen aussieht:

```
FOR x1=0 TO 19
  POKEW Ebnadr(b)+2*x1+40*y1,CVI(INPUT$(2,1))
NEXT x1
```

Nun werden immer nur noch 2 Bytes eingelesen, die mit CVI umgewandelt und mit POKEW in den Speicher gebracht werden. Die ganze Aktion wird damit zwar leider etwas langsamer, aber wenigstens läuft das Programm nun auch nach dem Compilieren, anstatt anzustürzen.

Der Befehl CVL kommt jedoch noch an zwei anderen Stellen im Programm vor. Einmal in dem Programmteil, der mit SUB Laden STATIC beginnt und einmal im Teil "Einlesen:". Bitte ersetzen Sie in beiden Programmteilen die Zeilen

```
Laenge=CVL(INPUT$(4,1))
```

jeweils durch die Zeile

```
Laenge=CVI(INPUT$(2,1))*65536+CVI(INPUT$(2,1))
```

Wir berechnen den 32-Bit-Wert aus zwei 16-Bit-Zahlen und vermeiden so den CVL-Befehl. Bevor Sie das Programm abspeichern und compilieren, sollten Sie es zunächst in AmigaBASIC testen. Dabei würden sich dann eventuelle Tippfehler zeigen.

- Wenn alles ordnungsgemäß funktioniert, speichern Sie die Version bitte mit

```
save "Videotitel.IFF",a
```

zurück auf Diskette und verlassen Sie dann AmigaBASIC.

- Kopieren Sie das Quellprogramm von Ihrer Arbeitsdiskette auf die RAM-Disk, und starten Sie dann AC/BASIC.

- Bitte compilieren Sie "ram:Videotitel.IFF". Vergessen Sie dabei nicht, die Option "R Link Run-time" anzuklicken.

- Kopieren Sie nach dem Compilieren "Videotitel.IFF.run" von der RAM-Disk auf Ihre Arbeitsdiskette.

Jetzt können Sie das compilierte Videotitel-Programm nach Herzenslust ausprobieren. Auch das Einlesen von Hintergrundbildern funktioniert nun einwandfrei. Eine typische IFF-Datei haben wir mit der compilierten Version in etwa 40 Sekunden eingelesen. Zum Vergleich: Die uncompilierte Ur-Version (in der 4 Bytes auf einmal gelesen werden konnten) benötigt 93 Sekunden.

Beim Ausprobieren können Sie auch gut testen, an welchen Stellen sich der Compiler bemerkbar macht. Im Menüpunkt 4 (Farben festlegen) zum Beispiel geht das Verschieben der Farbregler über die Tastatur deutlich schneller. Auch die Objektbewegung bei der Wiedergabe des Titels ist leicht beschleunigt.

Wir wollen aber nicht verschweigen, daß das Programm bei unseren Tests von Zeit zu Zeit trotzdem abstürzte. Besonders häufig war das dann der Fall, wenn wir fertige Titelsequenzen von Diskette eingelesen haben. Es ist uns nicht gelungen, einen spezifischen Fehler zu finden, der diese Abstürze verursacht. Offenbar hat die aktuelle AC/BASIC-Version beim Einlesen von Daten von Diskette grundsätzlich Schwierigkeiten.

In diesem Zusammenhang möchten wir aber darauf hinweisen, daß die Käufer von AC/BASIC von Absoft über einen Update-Service oder sogenannte Patches versorgt werden, wenn es neue verbesserte Versionen des Compilers gibt. Es ist sicher nur eine Frage der Zeit, bis Absoft die Fehler beim Einlesen von Diskette erkannt und behoben hat.

8.4 Sie haben die Wahl - Die Compiler-Optionen von AC/BASIC

Nachdem Sie jetzt Ihre ersten eigenen Erfahrungen mit AC/BASIC gemacht haben, wollen wir ins Haupt-Window des Compilers zurückkehren und Ihnen die verschiedenen Optionen erklären, die Ihnen beim Compilieren zur Verfügung stehen.

Die meisten Programme lassen sich mit der Einstellung, die wir Ihnen vorgeschlagen haben (Optionen C, N, R und T aktiv) problemlos compilieren. Sie sollten auf jeden Fall wissen, was diese Optionen bewirken. Wer sich näher mit AC/BASIC beschäftigen möchte, ist sicher auch an den anderen Möglichkeiten und ihrer Bedeutung interessiert. Allen Besitzern von AC/BASIC empfehlen wir jedoch unbedingt, die Informationen im Handbuch zu lesen, da dort noch einige wertvolle Tips und Erklärungen gegeben werden. Wir wollen in unserem Buch nur eine Übersicht über die Möglichkeiten geben.

Wie eine bestimmte Option an- bzw. ausgeschaltet wird, wissen Sie schon: Durch Klicken in das Kästchen vor einer Option schalten Sie den jeweiligen Punkt an. Zur Kontrolle wird das Kästchen orange ausgemalt. Ein weiterer Klick schaltet die Option wieder ab - das Kästchen ist wieder blau. Die Optionen bleiben solange eingestellt, wie Sie den AC/BASIC-Compiler benutzen. Sobald Sie ihn jedoch verlassen, vergißt er auch die eingestellten Optionen. Sie können aber durch einen Klick ins "SAVE"-Kästchen rechts unten im Haupt-Window die aktuellen Einstellungen auf der Compiler-Diskette abspeichern. Beim nächsten Start des Programms sind Ihre Optionen dann voreingestellt.

C Enable Run-time Tests

(Runtime-Tests durchführen) Wenn Sie diese Option anwählen, führt das compilierte Programm während des Ablaufs Fehlerkontrollen durch, so wie Sie das vom AmigaBASIC-Interpreter gewohnt sind. Tritt dann während des Programmablaufs ein Fehler auf, erscheint eine Fehlermeldung im aktuellen Ausgabewindow. Diese Fehlermeldungen sind im Gegensatz zu Amiga-

BASIC keine Klartext-Meldungen, sondern lediglich Fehlernummern. Die Bedeutung dieser Fehlernummern können Sie im AC/BASIC-Handbuch im Anhang F nachlesen. Der Nachteil dieser Fehlerüberprüfungen ist, daß sie die Verarbeitungsgeschwindigkeit eines Programms deutlich bremsen können. Wenn es Ihnen auf hohe Geschwindigkeit ankommt, sollten Sie diese Option deshalb abschalten.

N Process Run-time Events

(Runtime-Events verarbeiten). Bei dieser Option geht es um Event Trapping. Wenn ein Programm Event Trapping benutzt, müssen Sie diese Option anschalten. Der Compiler schreibt das Maschinenprogramm dann so, daß ständig die möglichen Events überprüft werden. Wenn Sie diese Option ausschalten, kann das Object-Programm kein Event Trapping durchführen. Wenn in Ihrem Programm OBJECT-Befehle vorkommen, muß die N-Option unbedingt aktiviert sein.

R Link Run-time

(Runtime-Library einbinden) Damit ein compiliertes Programm laufen kann, ist eine Bibliothek aus verschiedenen Maschinenroutinen nötig. Wir haben darüber schon kurz im Kapitel 8.2 gesprochen. In dieser sogenannten "Runtime-Library" stehen verschiedene Funktionen, die das Object-Programm aufrufen muß.

Sie haben nun zwei Möglichkeiten: Entweder Sie aktivieren die Option R (so wie wir das bisher immer gemacht haben) und binden damit die Runtime-Library automatisch in Ihr Programm mit ein. Der Vorteil dieses Systems ist, daß das compilierte Programm dann eigenständig und völlig unabhängig laufen kann. Der Nachteil ist der benötige Speicherplatz. Die Runtime-Library ist ca. 40 KByte lang und diese 40 KByte kommen zur Länge des erzeugten Maschinenspracheprogramms dazu. Ihre Object-Dateien werden daher ziemlich groß.

Die Alternative besteht darin, die R-Option auszuschalten. Sobald Ihr Maschinenprogramm geladen ist, sucht es dann im aktuellen L-Verzeichnis seine Runtime-Library. Dabei handelt es sich entweder um die Datei BAS.DL oder die Datei BAS.RL, je nachdem welche Mathe-Routinen Sie mit der Option D ausgewählt haben. Der Nachteil besteht darin, daß Sie dafür sorgen müssen, daß Ihr Programm die gesuchte Runtime-Datei in jedem Fall findet. Dafür sparen Sie aber auch wieder eine ganze Menge Speicherplatz auf Diskette. Wenn Sie mehrere compilierte Dateien auf einer Diskette haben, lohnt es sich, ein L-Verzeichnis anzulegen und die Runtime-Library dort hineinzukopieren.

T Temporaries on RamDisk

(temporäre Dateien auf der RAM-Disk anlegen) Während der Compiler Ihr Programm bearbeitet, legt er verschiedene temporäre Arbeitsdateien an. Diese Dateien werden nach dem Compilieren wieder automatisch gelöscht. Mit der T-Option können Sie AC/BASIC veranlassen, diese temporären Dateien auf der RAM-Disk anzulegen. Das Compilieren geht dann wesentlich schneller, weil die Zugriffszeiten auf die RAM-Disk viel kürzer sind als die auf eine Diskette oder die Festplatte. Schwierigkeiten kann es allerdings geben, wenn Sie nicht genug Speicherplatz zur Verfügung haben. Wenn AC/BASIC eine entsprechende Meldung bringt oder der Requester "Volume RAM is full" erscheint, müssen Sie Ihre temporären Dateien wohl oder übel auf eine Diskette auslagern und die T-Option abschalten.

A Use Long Addressing

(32-Bit-Adressierung verwenden) Hier geht es um die Anzahl an Bytes, die im Maschinenprogramm für die Angabe einer Speicheradresse verwendet werden. Im Normalfall lassen Sie diese Option einfach ausgeschaltet. Der Compiler arbeitet dann mit 16-Bit-Adressen.

Sollte sich während des Compilierens herausstellen, daß diese 16 Bit für die Adressierung nicht ausreichen, bringt der Compiler eine entsprechende Meldung:

```
Address out of 16 bit range - use A option
```

In diesem Fall schalten Sie die A-Option ein. Durch die Verwendung von 32-Bit-Adressen wird der Object-Code etwas größer.

D Compile for Decimal Math

(Dezimalmodus für Mathe-Berechnungen) Diese Option geht schon ziemlich ans Eingemachte. Mit ihr können Sie die interne Darstellung von Zahlen beeinflussen. AmigaBASIC benutzt normalerweise eine Binär-Darstellung. AC/BASIC bietet alternativ dazu eine Dezimal-Darstellung, die Sie mit der D-Option anwählen können. Wenn Sie sich für dieses Thema näher interessieren, möchten wir Sie auf den Anhang D des Commodore-AmigaBASIC-Handbuchs und auf den Anhang A des AC/BASIC-Handbuchs verweisen.

Im Normalfall sollten Sie diese Option ausgeschaltet lassen. Wenn Sie Binär-Darstellung verwenden, benötigt Ihr Programm die Runtime-Library BAS.RL. Bei Dezimal-Darstellung ist es die Datei BAS.DL.

E Generate Errors List

(Fehler-Datei erzeugen) Mit dieser Option beauftragen Sie AC/BASIC, die Fehlermeldungen, die in Pass 1 auftreten, nicht auf den Bildschirm, sondern in eine Datei zu schreiben. Wenn in Ihrem Programm sehr viele Fehler auftreten, ist die Datei übersichtlicher und kann mehrfach durchgelesen werden.

I List Include Statements

(Include-Befehle auflisten) Wenn Sie einige Erfahrung mit AC/BASIC gesammelt haben, können Sie weitergehende Möglichkeiten anwenden. Dazu gehört z.B. das Einbinden von anderen BASIC-Programmen während des Compilierens. Der dafür zuständige Befehl heißt INCLUDE ("Einfügen") und wird im AC/BASIC-Handbuch beschrieben. Mit dieser Option können Sie die INCLUDE-Befehle in eine List-Datei ausgeben lassen. Wenn beim Compilieren Probleme auftreten, können diese Informationen hilfreich sein.

L Generate Full List

(komplettes Listing erzeugen) Wenn Sie diese Option anklicken, erzeugt der Compiler ein komplettes Listing Ihres Quell-Programms. Dieses Listing wird in einer Datei abgelegt, die den Namen Ihres Quell-Programms um ".lst" erweitert hat. Wenn Fehler auftreten, druckt AC/BASIC sie direkt unter die betroffene Zeile.

S Generate Symbol File

(Variablendatei erzeugen) Diese Option wird erst in Zukunft interessant, wenn Absoft möglicherweise Hilfprogramme zum Debugging anbietet.

U Default Arrays to Static

(Nur statische Datenfelder verwenden) Im Gegensatz zum AmigaBASIC-Interpreter kann AC/BASIC Datenfelder statisch verwalten. Das bedeutet, diese Datenfelder können nicht mehr gelöscht oder neu dimensioniert werden. Dadurch werden dem Object-Code komplizierte Berechnungen erspart, wodurch die Programme unter Umständen deutlich schneller laufen. Näheres dazu finden Sie im AC/BASIC-Handbuch.

Das waren sie, die Compiler-Optionen von AC/BASIC. Zusätzlich gibt es auch noch die Möglichkeit, ins Programmlisting sogenannte "Metacommands" einzufügen, die bestimmte Optionen für einzelne Bereiche Ihres Source-Codes steuern. Diese Möglichkeiten führen jedoch für dieses Kapitel zu weit. Im Handbuch zu AC/BASIC sind sie alle ausführlich beschrieben.

In den folgenden Kapiteln wollen wir Ihnen nun noch einige Tips zum Compilieren einiger Programme aus diesem Buch geben.

8.5 Der schnelle Pinsel - Compilieren des Malprogramms

Wenn Sie unser AmigaBASIC-Malprogramm compilieren wollen, gibt es eigentlich nur eine größere Schwierigkeit zu umschiffen. Und selbst die haben Sie schon kennengelernt. Problematisch ist in der aktuellen Version von AC/BASIC der Fehler beim Befehl CVL, der die IFF-Leseroutine zum Absturz bringt.

Laden Sie also bitte das Malprogramm in AmigaBASIC, geben Sie

```
list BodyMap:
```

ins BASIC-Window ein und verändern Sie die Programmzeilen

```
FOR x1=0 TO 9
  POKEL Ebnadr(b)+4*x1+40*y1,CVL(INPUT$(4,1))
NEXT x1
```

so, daß sie folgendermaßen aussehen:

```
FOR x1=0 TO 19
  POKEW Ebnadr(b)+2*x1+40*y1,CVI(INPUT$(2,1))
NEXT x1
```

Wie schon im Videotitel-Programm, kommt auch im Malprogramm der Befehl CVL an noch zwei anderen Stellen vor. Nämlich im Teil "Laden:" und im Teil "Einlesen:". Bitte ersetzen Sie in diesen beiden Programmteilen die Zeilen

```
Laenge=CVL(INPUT$(4,1))
```

jeweils durch die Zeile

```
Laenge=CVI(INPUT$(2,1))*65536+CVI(INPUT$(2,1))
```

Auch beim Abspeichern von IFF-Grafiken müssen Sie etwas verändern. Im Programmteil "Speichern:" finden Sie die Zeile

```
OPEN Nam$ FOR OUTPUT AS 1 LEN = FRE(0)-500
```

Mit dieser Zeile haben wir in unserem Programm den verbleibenden BASIC-Speicher bis auf 500 "Sicherheits-Bytes" als Ausgabepuffer für unsere Datei genutzt. Die Funktion FRE(0) lieferte uns dabei den noch nicht belegten BASIC-Speicher. Bei compilierten Programmen gibt es keinen BASIC-Speicher. AC/BASIC liefert bei FRE(0) und FRE(-1) immer denselben Wert, nämlich den verbleibenden Systemspeicher. Wenn Sie aber versuchen, Speicherbereiche in der Größenordnung von mehreren 100K als Ausgabepuffer zu verwenden, erhalten Sie vom compilierten Programm einen "**** Error 6", was einem "Overflow"-Error entspricht.

Die einfachste Methode, dieses Problem zu umgehen, besteht darin, die LEN-Eingabe einfach zu löschen. Übrig bleibt in der genannten Zeile:

```
OPEN Nam$ FOR OUTPUT AS 1
```

Nach diesen Änderungen sollte die compilierte Version problemlos laufen. Deutliche Geschwindigkeitsverbesserungen merken Sie besonders beim Aufbau der Windows für "Farben ändern" und "Muster ändern" und den Einstellungen in diesen

Windows. Auch das Einlesen und Abspeichern von IFF-Grafiken geht schneller als in der uncompilierten Version. Viel Spaß beim Ausprobieren!

8.6 Der schnelle Blick - das Balken- und Tortengrafik-Utility

Ob sich das Compilieren unserer Statistikdaten-Verwaltung lohnt, darüber könnte man schon geteilter Meinung sein. Im überwiegenden Teil der Zeit ist das Programm mit Diskettenoperationen oder dem Erstellen von Grafiken beschäftigt. Deutliche Geschwindigkeitsunterschiede dürfte man höchstens im Editor erwarten. Da beim Compilieren dieses Programms aber einige typische Probleme auftreten, wollten wir es doch besprechen.

Mit dem SUB-Programm "PicSave" gibt es erfreulicherweise keine Probleme, da es sowieso schon am Programmende steht. Sie müssen lediglich im Quell-Programm in der "PicSave"-Routine die Zeile

```
OPEN Nam$ FOR OUTPUT AS 1 LEN = FRE(0)-500
```

gegen die Zeile

```
OPEN Nam$ FOR OUTPUT AS 1
```

ersetzen. Dieses Problem und seine Lösung kennen Sie schon vom Malprogramm. Beim ersten Versuch, dieses Programm zu compilieren, meldet Pass 1 jedoch zwei Fehler:

```
    RUN

***** error in line 00290 [   RUN]: 43 - Unexpected end of statement

    IF FeldJN%=1 THEN SHARED Farben%()

***** error in line 00362 [HARED ]: 16 - Illegal statement ordering
```

Beide Fehler hängen wieder mit Eigenheiten von AC/BASIC zusammen. Zum einen erlaubt AC/BASIC den RUN-Befehl nicht in der Form, wie wir den Befehl benutzt haben. Der Compiler versteht bloß die Form RUN (Programmname), die er dazu nutzt, ein anderes Programm nachzuladen. Wir benutzen den RUN-Befehl im Programmteil "DatenLoeschen:", weil er der schnellste Weg ist, den Speicher zu löschen und das Programm neu zu starten. Für den Compiler müssen wir diesen RUN-Befehl durch die Zeile

```
CLEAR : GOTO Vorbereitungen
```

ersetzen. Der zweite Fehler tritt innerhalb der "PicSave"-Routine am Programmende auf. Aufgrund der Organisation von Datenfeldern unter AC/BASIC erlaubt der Compiler den SHARED-Befehl nicht hinter IF...THEN-Ausdrücken oder anderen Bedingungen. In diesem Programm ist das nicht weiter schlimm. Sie können die Zeile

```
IF FeldJN%=1 THEN SHARED Farben%()
```

ersatzlos löschen. Denn die "PicSave"-Routine wird vom Hauptprogramm sowieso mit dem Parameter FeldJN%=0 aufgerufen. Das heißt, das SUB-Programm soll seine eigene Farbtabelle anlegen und zum Abspeichern verwenden. Da in den folgenden Programmzeilen ein ERASE-Befehl vorkommt, dürfen Sie das Statistikdaten-Programm nicht mit eingeschalteter U-Option ("Default Arrays to STATIC") compilieren.

Die Unterschiede in der Behandlung von Datenfeldern zwischen dem Interpreter und dem Compiler machen es außerdem notwendig, daß Sie im Programmteil "Vorbereitungen:" auch das Feld 'Farben%' dimensionieren. Und zwar im SHARED-Modus, also auch für Verwendung in SUB-Programmen. AmigaBASIC hat diese Dimensionierung bei der ersten Verwendung des Felds automatisch vorgenommen. Das tut AC/BASIC nicht. Die ersten drei Zeilen des Listings müssen also so aussehen:

```
Vorbereitungen:
  DIM Nummer$(58),Bez$(58),Zahl$(58)
  DIM SHARED Farben%(31,2)
```

Das Programm läuft nach dem Compilieren problemlos. Lediglich eine Schwierigkeit bei der Bedienung ergibt sich: AC/BASIC scheint im Gegensatz zum AmigaBASIC-Interpreter ein Window durch einen WINDOW-Befehl nicht automatisch in den Vorgrund zu holen. Wenn eine Grafik aufgebaut ist und Sie durch Tastendruck zurück ins Daten-Window gelangen wollen, wird das Daten-Window zwar zum aktuellen Window erklärt, aber es bleibt hinter dem Screen mit der Grafikdarstellung verborgen.

Ein Trick, wie man sich aus dieser Situation befreien kann, ist, die Pulldown-Option "Datei laden" anzuwählen. Nun erzeugt das Programm ein neues Window (in dem Sie den Dateinamen eingeben sollen), wodurch der Screen mit dem Daten-Window nach vorne geholt wird. Wenn Sie dann im Window "Bitte Dateinamen eingeben:" einfach eine leere Eingabe machen, also nur <RETURN> drücken, bleiben Ihre Daten im Speicher und Sie sind wieder auf dem richtigen Screen gelandet. Wem dieser Kniff auf Dauer nicht gefällt, kann auch im Quell-Programm den letzten Block des Programmteils folgendermaßen erweitern:

Vorher:

```
      GOSUB Grafik

      WINDOW 2,"Bitte drücken Sie eine Taste!",(350,0)-(631,0),20,4
      COLOR 0,1 : CLS
      WHILE INKEY$=""
      WEND
      WINDOW CLOSE 2
      WINDOW 1
      MENU ON
      MENU 1,0,1 : MENU 2,0,1
    END IF
  RETURN
```

Nachher:

```
        GOSUB Grafik

        WINDOW 2,"Bitte drücken Sie eine Taste!",(350,0)-(631,0),20,4
        COLOR 0,1 : CLS
        WHILE INKEY$=""
        WEND
        WINDOW CLOSE 2
        WINDOW 2," Dummy",(0,0)-(0,0),0,-1
        WINDOW CLOSE 2
        WINDOW 1
        MENU ON
        MENU 1,0,1 : MENU 2,0,1
      END IF
    RETURN
```

Wenn Sie das Programm danach neu compilieren, sind Sie auch das Problem mit den Windows los. Dadurch, daß wir auf dem Screen -1 (dem Workbench-Screen) ein winziges Window erzeugen und gleich wieder löschen, bringt auch AC/BASIC den richtigen Screen in den Vordergrund.

8.7 Die schnellen Wellen - das Synthesizer-Utility

Zu guter Letzt wollen wir noch zusammen das Synthesizer-Utility compilieren. Vielleicht fragen Sie sich, warum wir das Datenbank-Programm und das Sprachutility ausgelassen haben. Ganz einfach: Erinnern Sie sich bitte daran, wann es sich lohnt, ein Programm zu compilieren. Na? Beim Datenbankprogramm dreht sich alles um relative Dateiverwaltung. Die Geschwindigkeit hängt fast nur von der Zugriffsgeschwindigkeit der Diskettenlaufwerke bzw. der Festplatte ab. Also wenig Verbesserungsmöglichkeiten für den Compiler. Ähnlich beim Sprachutility. Da ist das einzige verzögernde Moment der Grafikaufbau. Und der kann bekanntlich auch nicht schneller gemacht werden.

Anders sieht's nun wieder beim Synthesizer-Utility aus. Besonders der Wellenaufbau kostet viel Rechenzeit. Hier müßte der Compiler einiges herausschlagen können. Und um es Ihnen gleich zu verraten: Er tut es auch. Bei unseren Tests mit diesem Programm, trat ein merkwürdiges Problem auf. Obwohl im Quell-Programm klar und eindeutig stand, daß beim Abspeichern auf Diskette 256 Werte geschrieben werden sollen, wurden die abgespeicherten Dateien immer nur 255 Werte lang. Versuchte man dann, diese Datei wieder einzulesen, fehlte der Leseroutine natürlich ein Wert und das Programm brach mit **** Error 62 - "Input past End" ab.

Nachdem wir dann testhalber die Lese- und Schreibroutine auf 255 Werte verkleinerten, lief alle wie am Schnürchen. Möglicherweise hat AC/BASIC bei der Verwaltung der Zahlen ein Problem. Jedenfalls haben wir keine andere Erklärung gefunden. Da sich der Fehler auf keine andere Art beheben ließ, empfehlen wir Ihnen, die Schleifen einfach auf 255 zu kürzen. Der letzte Wert der abgespeicherten Wellenformen kann vernachlässigt werden. Ändern Sie also bitte die Programmzeilen

```
OPEN Name$ FOR OUTPUT AS 1
  FOR x=0 TO 256
    PRINT #1,CHR$(127-Welle%(x));
  NEXT x
CLOSE 1
```

in

```
OPEN Name$ FOR OUTPUT AS 1
  FOR x=0 TO 255
    PRINT #1,CHR$(127-Welle%(x));
  NEXT x
CLOSE 1
```

und den Block

```
OPEN Name$ FOR INPUT AS 1
  FOR x=0 TO 256
    Welle%(x)=127-ASC(INPUT$(1,1))
  NEXT x
CLOSE 1
```

in

```
OPEN Name$ FOR INPUT AS 1
  FOR x=0 TO 255
    Welle%(x)=127-ASC(INPUT$(1,1))
  NEXT x
CLOSE 1
```

Sonst brauchen Sie beim Compilieren des Synthesizer-Utilities nichts weiter zu beachten. Wenn Sie die compilierte Version testen, wird Ihnen sicher auffallen, daß die Berechnung und Darstellung der Wellenformen im Vergleich zur Interpreter-Version stark beschleunigt ist. Wie gesagt - bei rechenintensiven Programmen lohnt sich das Compilieren.

Tja, das war's. Mit dem Compilieren des Synthesizer-Utilities ist nun auch das letzte Kapitel in unserem Buch zuende gegangen. Wenn sich der eine oder andere von Ihnen nun entschlossen hat, sich den AC/BASIC-Compiler zu kaufen oder einige ihn vielleicht schon besitzen, wollen wir Ihnen an dieser Stelle nochmal ans Herz legen, das Compiler-Handbuch durchzulesen. Viele Details und Besonderheiten konnten wir gar nicht in der Ausführlichkeit beschreiben, wie Sie sie in der Absoft-Dokumentation finden.

Übrigens: Einige Funktionen wollten wir auch nicht beschreiben - denn dieses Kapitel und die compilierten Versionen der Programme in der Compiler-Schublade, sollten keine Hilfestellung für Raubkopierer sein! Wir wollten denen die dieses Programm gekauft haben und trotzdem nicht so ganz zurecht kommen - sei es wegen des englischen Handbuches oder auch

wegen der Komplexität des Themas - mit ein paar Hinweisen und Tricks helfen. Der letzte Tip: Besonders der Referenzteil des Compiler-Handbuches, in dem für jeden AmigaBASIC-Befehl die Besonderheiten des Compilers erklärt werden, ist beim Austesten der Programme Gold wert.

Es folgen nun noch einige für Sie nützliche und informative Anhänge, aber vorher müssen wir uns natürlich noch schnell verabschieden. Und bei der Menge an Büchern, die wir mittlerweile geschrieben haben, ist es sozusagen eine Abschieds-und-Aufwiedersehens-Party Alle Gäste dieser Veranstaltung sind im nächsten Minikapitel herzlich willkommen.

Das war's



Leider haben wir gerade eben erfahren, daß man so kurzfristig (von einer Seite zur nächsten) weder Champagner noch andere Delikatessen auftreiben kann. Also muß unser Abschied mit den bescheidenen Mitteln des Wortes stattfinden.

Zuallererst hoffen wir natürlich, daß sie Ihnen ein wenig Spaß gemacht hat, unsere Reise durch die Welt des AmigaBASIC. Wenn Sie jetzt Lust haben, selber in BASIC Programme zu schreiben, dann haben wir unser Ziel eigentlich schon erreicht. Übrigens: Wir würden uns freuen, wenn Sie auch mal zur Feder greifen. Es müssen ja nicht gleich 700 Seiten dabei herauskommen. Aber eine kurze Kritik (es darf natürlich auch ein langes Lob sein) würde uns schon freuen. Dazu aber gleich auch ein wichtiger Hinweis: Der Kontakt zu unseren Lesern ist uns wichtig. Egal ob auf Messen, per Post oder wenn es gar nicht anders geht, per Telefon. Aber zwei Anmerkungen: Wir bitten um Verständnis, wenn die Antwort auf Ihren Brief ein wenig auf sich warten läßt. Sie schreiben einen an uns, aber wir beantworten eine ganze Menge Briefe und das braucht Zeit - zumal wir zwischendrin ja auch noch hie und da ein Buch schreiben...

Zweitens: Wenn man unsere Telefonnummern hat und uns anruft, geben wir durchaus gerne Auskunft. Aber wie jeder andere normale Mensch sind wir von Anrufen an Sonn- und Feiertagen, in aller Hergottsfrühe oder gar mitten in der Nacht nicht so begeistert. Das ist durchaus nicht böse gemeint, aber ein bißchen Privatleben gehört halt nun mal auch dazu. Denn damit erhalten wir uns die gute Laune, die vielen von Ihnen das Lesen unserer Bücher so angenehm macht.

Ansonsten wünschen wir Ihnen auch weiterhin viel Spaß mit Ihrem Amiga. Und wer weiß, vielleicht sehen wir uns mal wieder - in einem anderen Buch.

Vorerst bleibt uns nur noch, einige Leute dankend zu erwähnen, die mehr oder weniger wesentlich zum Gelingen dieses Buches beigetragen haben.

Ganz wichtig ist hier Alexander Graham Bell. Seiner genialen Erfindung (dem Telefon) haben wir es besonders zu verdanken, daß trotz der räumlichen Trennung zwischen Hannes (Würzburg/Stuttgart) und Christian (Düsseldorf) ein gemeinsames Buch entstanden ist. Der Deutschen Bundespost danken wir nicht, denn die hat ja die Gebühren dafür kassiert.

Toleranz zu bescheinigen ist Brigitte, die nicht nur hie und da tröstende Worte in den heißen Phasen während unserer letzten langen Nächte in Düsseldorf fand ("Willst du ein Bonbon?"), sondern auch nicht böse über all die viele Zeit war, die ihr und Christian auf diese Weise entging. (Der hatte nämlich Urlaub - eigentlich...) Und diesen Mut behielt sie erfreulicherweise bei der ersten, zweiten, dritten und jetzt der vierten Auflage des Buches.

Als nächstes gilt unser Dank Hannes' Eltern, die sich irgendwie daran gewöhnt hatten, ihren Sohn mehrere Monate lang nur beim Essen zu Gesicht zu bekommen.

Ganz besonders danken wir auch Herrn Dr. Peter Kittel von Commodore Frankfurt für die freundliche Unterstützung.

Ein großes Dankeschön geht auch nach Westchester/USA an Carolyn Scheppner, die uns erlaubte, Teile ihres Programms "LoadILBM-SaveACBM" für unser "ILBMEntzerren" zu verwenden. Hi Carolin, Thanks a lot!

Und weil wir gerade dabei sind: Den Softwarehäusern Electronic Arts, Discovery Software und Gold Disk verdanken wir die Entstehung der Skizzen und Bildschirmausdrucke. Die Skizzen wurden mit dem Programm Deluxe Paint II gemacht, die Hardcopies haben wir mit "GRABBit" abgespeichert, und die Ausdrucke auf unserem Laserdrucker mit "Professional Page" erstellt.

Hannes dankt noch Johannes - es lebe die Musik, die lange Nächte erträglich macht.

Bleibt noch Vater Staat (vertreten durch die Bundeswehr Veitshöchheim), der dem Gefreiten Rügheimer hie und da kurzfirstig ein paar Tage Urlaub gönnte, um die erste Auflage dieses Buch zu schreiben.. (Daß er dafür wieder Steuern kriegt, steht ja auf einem anderen Blatt...)

IV. Anhänge

Anhang A: Fehlermeldungen und Abhilfe

In diesem Anhang sind alle Fehlermeldungen, die bei der Arbeit mit AmigaBASIC auftreten können, alphabetisch aufgeführt. Wenn beim Arbeiten mit diesem Buch oder beim Programmieren ein Fehler auftritt, können Sie hier nachschlagen, was die möglichen Ursachen sind und was man gegebenenfalls dagegen unternehmen kann.

Ein Teil der Fehler wird von AmigaBASIC schon vor dem Programmablauf entdeckt, ein Teil erst unmittelbar vor der Ausführung des betroffenen Befehls. In jedem Fall wird der Fehler im LIST-Window markiert, und ein Error-Dialogwindow erscheint. Bevor Sie irgend etwas gegen den Fehler unternehmen können, müssen Sie erst ins OK-Feld der Meldung klicken. Die Fehlernummern, die in dieser Übersicht aufgeführt sind, brauchen Sie im Zusammenhang mit dem Befehl ON ERROR und der Systemvariablen ERR. Näheres darüber im Anhang B.

Advanced Feature
Fehlernummer: 73
"Eigenschaft eines erweiterten BASIC"

Wenn Sie diesen Fehler erhalten, haben Sie einen Befehl verwendet, der nicht in unserem Buch vorkommt. AmigaBASIC bringt diese Meldung, wenn es einen Befehl zwar erkennt, aber in der gegenwärtigen Version noch nicht ausführen kann. Uns ist bisher noch kein solcher Befehl bekannt. Dieser Fehler ist ein Überbleibsel der anderen BASIC-Versionen von Microsoft. Beim IBM PC gibt es z.B. drei verschieden leistungsfähige Microsoft-BASIC-Versionen.

Argument count mismatch
Fehlernummer: 37
"Fehler in der Anzahl an Argumenten"

Tritt dann auf, wenn Sie beim Aufruf eines SUB-Programms zu viele oder zu wenige Argumente (Werte, Texte) angeben. Schauen Sie in der ersten Zeile des SUB-Programms nach, wieviele und welche Angaben verlangt werden, und geben Sie genauso viele im Aufruf an.

Bad file mode
Fehlernummer: 54
"Falscher Datei-Modus"

Dieser Fehler tritt dann auf, wenn Sie bei der Arbeit mit Dateien Befehle oder Angaben verwenden, die nicht zur angegebenen Datei passen. Im einzelnen kann es folgende Ursachen geben:

- Sie wollen mit MERGE ein Programm anhängen, das nicht im ASCII-Format gespeichert wurde. In diesem Fall müssen Sie das Programm laden und mit SAVE "(Programmname)",A erneut abspeichern.

- Sie wollen eine relative Datei mit LOAD laden. Das geht natürlich nicht.

- Sie verwenden für eine sequentielle Datei einen der Befehle für relative Dateien, wie GET oder PUT.

- Sie haben für den OPEN-Befehl einen anderen Modus als "I", "O", "A" oder "R" angegeben. Andere Modi gibt es aber nicht.

Bad file name
Fehlernummer: 64
"Falscher Dateiname"

Der Dateiname, den Sie angegeben haben, ist falsch. Das heißt, er ist entweder zu lang oder nicht so aufgebaut, wie von Amiga-BASIC erwartet. In Kapitel 3.2 erfahren Sie, wie ein zulässiger Dateiname aussieht. Er hat im Höchstfall folgende Bestandteile:

"(Gerätename oder Laufwerk):(Schublade)/(Schublade)/.../(Dateiname)"

Bad file number
Fehlernummer: 52
"Falsche Dateinummer"

Bedeutet fast immer, daß Sie bei einem Befehl wie PRINT# oder INPUT# die Nummer einer Datei angegeben haben, die nicht geöffnet wurde. Schauen Sie sich die OPEN-Befehle in Ihrem Programm an, da stehen die Dateinummern.

Bad record number
Fehlernummer: 63
"Falsche Satznummer"

Bei einem GET- oder PUT-Befehl wollen Sie einen Satz angeben, der nicht erlaubt ist. Zulässige Satznummern liegen zwischen 0 und 16777215.

Block ELSE/END IF must be 1st statement on the line
Fehlernummer: -
"ELSE oder END IF in einem IF/END IF-Block muß der erste Befehl in der Zeile sein."

Die beiden müssen sogar der erste und der einzige Befehl in der Zeile sein. Sonst erkennt AmigaBASIC nicht, daß es sich um eine IF/ELSE/END IF-Struktur handelt. Strukturierte Programmierung hilft, solche Probleme zu verhindern. Dieser Fehler wird schon vor der Programmausführung erkannt und angezeigt.

Can't continue
Fehlernummer: 17
"Ich kann nicht weitermachen"

Tritt nach der Eingabe von CONT oder dem Anwählen von "Continue" aus dem "Run"-Pulldown auf: Weitermachen ist nur möglich, wenn kein Fehler aufgetreten ist, und das Programm seit dem Abbruch nicht verändert wurde. Sie müssen das Programm in diesem Fall mit RUN neu starten.

Deadlock
Fehlernummer: 77
"Verriegelung (!?)"

Hier sind wir uns selbst nicht ganz sicher. Diese Fehlermeldung gibt es, aber es sind bisher keine Informationen darüber aufzutreiben, wer oder was da verriegelt wird. Wahrscheinlich kommt der Fehler nicht von AmigaBASIC, sondern direkt vom Betriebssystem. Er klingt auf jeden Fall sehr unerfreulich, etwa in der Art von 'Internal error'. Sollten Sie diesen Fehler erhalten, schreiben Sie's uns doch mal.

Device I/O error
Fehlernummer: 57
"Gerätefehler bei Ein- oder Ausgabe"

Bei einem Input/Output-Vorgang hat AmigaDOS einen ernsten Fehler festgestellt. Ein technisches Problem, z.B. mit einer Diskette, könnte die Ursache sein. Meist müssen Sie in so einem Fall die Workbench neu laden.

Device Unavailable
Fehlernummer: 68
"Gerät ist nicht verfügbar"

Das angesprochene Gerät ist entweder nicht angeschlossen oder nicht eingeschaltet. Es ist AmigaBASIC nicht gelungen, Verbindung mit dem Gerät aufzunehmen. Da AmigaBASIC besonders bei der Arbeit mit Peripheriegeräten hin und wieder Fehler macht, müssen Sie bei dieser Meldung möglichweise die Workbench neu laden.

Disk full
Fehlernummer: 61
"Diskette ist voll"

Tritt meistens beim Speichern von Programmen oder Schreiben von Daten auf: Auf der Diskette ist nicht mehr genug Platz. Entweder Sie löschen ein paar Programme bzw. kopieren sie auf eine andere Diskette, oder Sie verwenden eine neue Diskette für die Speicherung, die den Fehler verursachte. Wenn Ihnen dieser Fehler zum ersten Mal begegnet, während Sie in den ersten Kapiteln unseres Buchs mit der Extras-Diskette arbeiten, lesen Sie bitte Kapitel 3.1.

Division by zero
Fehlernummer: 11
"Division durch Null"

Dividieren durch Null ist verboten. PRINT 1/0 erzeugt also z.B. diesen Fehler. Es kann auch daran liegen, daß eine Variable noch keinen Wert hat oder daß in Ihren Formeln ein Fehler steckt. Oder haben Sie sich vielleicht beim Variablennamen vertippt?

Duplicate Definition
Fehlernummer: 10
"Doppelte Dimensionierung"

Dieser Fehler tritt auf, wenn Sie ein bereits dimensioniertes Datenfeld nochmal dimensionieren wollen. Daß das Feld bereits dimensioniert ist, kann verschiedene Gründe haben: Entweder es gab schon einmal einen DIM-Befehl für denselben Variablennamen. Oder Sie haben das Feld schon einmal verwendet, bevor das zugehörige DIM im Programm steht. Dadurch wird ein Feld ja automatisch dimensioniert. Sie können in einem solchen Fall ein Feld mit ERASE löschen. Mit LBOUND und UBOUND erhalten Sie die Ober- und die Untergrenze des Datenfelds. Wenn Sie den Befehl OPTION BASE nach dem ersten DIM im Programm verwenden, kommt ebenfalls diese Fehlermeldung.

Duplicate label
Fehlernummer: 33
"Doppeltes Label, doppelte Sprungmarke"

Im selben Programm haben Sie für zwei verschiedene Programmteile dasselbe Label verwendet. Einen Programmteil müssen Sie umtaufen! Oder erkennt AmigaBASIC vielleicht eine Labeldefinition, wo gar keine beabsichtigt war? Dann muß der Doppelpunkt am Ende des Texts weg. Vielleicht liegt es an einer Verwechslung von SUB-Programm und Unterprogramm? Sie dürfen zwar einem SUB-Programm einen Namen geben, der schon als Label verwendet wird (obwohl das nicht zu empfehlen ist, da es zu Unklarheiten führt), aber ein klärendes CALL beim Aufruf ist dann in Zweifelsfällen schon nötig.

ELSE/ELSE IF/END IF without IF
Fehlernummer: -
"ELSE/ELSE IF/END IF ohne zugehöriges IF"

Einer der genannten Befehle wurde gefunden, ohne daß Sie mit IF einen gültigen IF...END IF-Block begonnen haben. Haben Sie vielleicht in der IF-Zeile hinter THEN weitere Befehle einge-

geben? Dürfen Sie nämlich nicht, wenn sich's um einen IF...END IF-Block handelt, denn dadurch findet ja die Unterscheidung statt. Diesen Fehler erkennt AmigaBASIC schon vor dem Programmstart.

EXIT SUB outside of a subprogram
Fehlernummer: -
"EXIT SUB außerhalb eines SUB-Programms"

Den Befehl EXIT SUB dürfen Sie nur innerhalb der Klammer SUB/END SUB verwenden. Ist das END SUB zu weit nach oben gerutscht oder haben Sie END SUB und EXIT SUB verwechselt? Auch diesen Fehler merkt AmigaBASIC sofort, wenn es sich Ihr Programm vor dem Start kurz ansieht.

FIELD overflow
Fehlernummer: 50
"Überlauf des Dateipuffers durch FIELD-Befehl"

Dieser Fehler bedeutet, daß bei einer relativen Datei mehr Pufferbytes mit FIELD angelegt werden sollen, als bei der Dateilänge im OPEN-Befehl angegeben wurde. Am besten, Sie planen Ihren Dateiaufbau ganz genau vor, dann wissen Sie immer, wieviele Bytes wofür verwendet werden.

File already exists
Fehlernummer: 58
"Datei existiert bereits"

Beim Speichern (SAVE) oder Umbenennen (NAME) haben Sie einen Namen angegeben, den es bereits gibt. Verwenden Sie einfach einen anderen Namen. Wenn es unbedingt der gewünschte Dateiname sein muß, wechseln Sie mit CHDIR in ein anderes Inhaltsverzeichnis.

File already open
Fehlernummer: 55
"Datei bereits geöffnet"

Tritt auf, wenn Sie versuchen, eine bereits geöffnete Datei nochmal zu öffnen. AmigaBASIC bringt diese Fehlermeldung auch, wenn Sie versuchen, eine zur Zeit geöffnete Datei mit KILL zu löschen. Vorher müssen Sie die Datei nämlich wieder schließen.

File not found
Fehlernummer: 53
"Datei nicht gefunden"

Die Datei, sie Sie angegeben haben, konnte nicht gefunden werden. Haben Sie sich beim Dateinamen vertippt? Oft befinden Sie sich auch im falschen Inhaltsverzeichnis (prüfen Sie's mit FILES und ändern Sie's gegebenenfalls mit CHDIR).

FOR without NEXT
Fehlernummer: 26
"FOR ohne NEXT"

AmigaBASIC hat einen FOR-Befehl gefunden, zu dem kein NEXT gehört. Entweder Sie haben das NEXT vergessen oder beim Namen der Zählvariablen einen Fehler gemacht. Am sichersten ist es, FOR...NEXT-Schleifen schön übersichtlich strukturiert zu programmieren und hinter NEXT gar keinen Variablennamen mehr anzugeben. (Das ist nämlich auch erlaubt.)

IF without END IF
Fehlernummer: -
"IF-Befehl ohne END IF"

Wenn Sie einen IF/END IF-Block verwenden, muß er mit END IF aufhören. Haben Sie END IF vielleicht vergessen? Oder wollten Sie gar keinen IF/END IF-Block erzeugen, haben aber

hinter THEN in der IF-Zeile nichts mehr geschrieben? Dann denkt AmigaBASIC nämlich, es handle sich um einen IF/END IF-Block. Wollen Sie hinter einem THEN keinen Befehl angeben, so schreiben Sie am besten ein REM dorthin. Weil dieser Fehler vor der Programmausführung entdeckt wird, kann er beim Entwickeln eines Programms auftreten, weil Sie eine Zeile noch nicht zu Ende programmiert haben. Vor der Ausführung von Befehlen im Direktmodus überprüft AmigaBASIC das Programm im LIST-Window nämlich auf offensichtliche Fehler. 'IF without END IF' ist einer davon. Sie erkennen solche Fehler in dieser Übersicht daran, daß sie keine Fehlernummer haben.

Illegal direct
Fehlernummer: 12
"Unerlaubter Direktbefehl"

Manche Befehle sind nur innerhalb von Programmen erlaubt und nicht im Direktmodus. Den obenstehenden Fehler erhalten Sie, wenn Sie so einen Befehl im BASIC-Window eintippen. DEF FN, COMMON oder SUB sind typische Beispiele.

Illegal function call
Fehlernummer: 5
"Unerlaubter Funktionsaufruf"

Dieser Fehler tritt auf, wenn Sie einen unerlaubten Wert an einen BASIC-Befehl übergeben. Wenn Sie negative Werte für Datenfeld-Elemente oder GET- und PUT-Befehle angeben, kommt er ebenfalls. Auch wenn Sie zu große, zu kleine oder falsche Parameter bei BASIC-Befehlen, die verschiedene Optionen haben, angeben, kann der Fehler entstehen. Tritt diese Fehlermeldung nach LIST auf, wurde das BASIC-Programm geschützt abgespeichert (SAVE "...",P).

Input past end
Fehlernummer: 62
"Einlesen hinter dem Dateiende versucht"

Kommt vor, wenn Sie Daten aus einer Datei lesen wollen, die bereits zu Ende ist oder die zum Schreiben (FOR OUTPUT) geöffnet wurde. Benutzen Sie die Befehle EOF und LOF, um diesen Fehler zu vermeiden.

Internal error
Fehlernummer: 51
"Interner Fehler"

Da können Sie wahrscheinlich gar nichts dafür. AmigaBASIC hat bei sich selbst einen Fehler gefunden. In diesem Fall müssen Sie AmigaBASIC leider neu starten.

Line buffer overflow
Fehlernummer: 23
"Zeilenpuffer ist voll"

Dieser Fehler tritt auf, wenn der Bildschirmeditor im LIST-Window Probleme hat. Vielleicht versuchen Sie, zuviele Zeichen in einer Zeile einzugeben? Nach 255 ist nämlich Schluß. Das Problem kann aber auch durch einen Fehler im Bildschirmeditor entstehen. Benutzen Sie ein paarmal CUT und PASTE aus dem "Edit"-Menü. Oft verschwindet der Fehler dann wieder.

Missing operand
Fehlernummer: 22
"Fehlender Wert hinter logischer Verknüpfung"

Wenn hinter Befehlen (wie AND, OR, XOR,...) oder Rechenzeichen (wie +, -...) kein Wert mehr folgt, erscheint dieser Error.

Missing STATIC in SUB statement
Fehlernummer: -
"STATIC fehlt beim SUB-Befehl"

Zu SUB gehört immer STATIC. Wenn das immer der Fall ist, hätte man's eigentlich auch weglassen können, da haben Sie schon recht. Trotzdem muß das STATIC angegeben werden, und wenn's fehlt, meldet AmigaBASIC einen Fehler. Und zwar schon vor dem Programmstart.

NEXT without FOR
Fehlernummer: 1
"NEXT ohne zugehöriges FOR"

AmigaBASIC hat ein überzähliges NEXT gefunden. Das ist das logische Gegenteil von 'FOR without NEXT'. Schauen Sie bitte dort nach, wenn Sie weiteres wissen wollen.

No RESUME
Fehlernummer: 19
"Kein RESUME gefunden"

AmigaBASIC konnte innerhalb einer ON ERROR-Routine (Fehlerbehandlung) keinen RESUME-Befehl finden. Dieser Befehl gehört aber nun mal dazu. Er schließt das behandelnde Unterprogramm ab und läßt das Programm danach zurück an die Stelle des Fehlers springen.

Out of DATA
Fehlernummer: 4
"Keine DATAs mehr"

Beim Einlesen von DATA-Werten mit READ haben Sie das Ende der DATA-Liste erreicht. Es gibt keine DATAs mehr. Das sollte Ihr Programm selbst merken können! Geben Sie z.B. -1 als letzten DATA-Wert an und vergleichen Sie die gelesenen Daten mit dieser Zahl. Und vergessen Sie nicht, daß Sie mit RESTORE den DATA-Zeiger verändern können.

Out of heap space
Fehlernummer: 14
"Keine freier Systemspeicher mehr"

Der hat Sie sicher ganz schön erschreckt! Er erscheint in kräftigem Rot auf dem Bildschirm und sieht fast aus wie eine 'Guru Meditation'. Ist aber keine. Dieser Fehler wird von AmigaBASIC erzeugt und sagt uns, daß der System-Speicherplatz nicht ausreicht. Klicken Sie einfach mit der linken Maustaste, dann geht's weiter. Vielleicht hilft CLEAR? Wenn nicht, speichern Sie Ihr Programm, starten Sie die Workbench neu und klicken Sie AmigaBASIC an. Wenn es keine anderen Speicherplatz-Anwärter im Amiga gibt, reicht der Platz manchmal etwas länger. Denken Sie daran, alle überflüssigen Programme und Windows zu schließen.

Out of memory
Fehlernummer: 7
"Kein freier BASIC-Speicher mehr"

Dieser Fehler bedeutet, daß im BASIC-Speicherplatz kein Speicher mehr zur Verfügung steht. Das heißt aber nicht, daß notwendigerweise im ganzen System keiner mehr frei ist. Nur Ihr Programm und Ihre BASIC-Daten brauchen zusammen zuviel Platz. Auch hier hilft oft der CLEAR-Befehl. Lesen Sie auch mal den Teil über 'Out of heap space'.

Overflow
Fehlernummer: 6
"Überlauf, Zahl zu groß"

Das bedeutet, daß eine Zahl zu groß ist, um im gewünschten Zahlenformat (siehe Zwischenspiel 7) dargestellt zu werden. Möglicherweise können Sie diesen Fehler vermeiden, wenn Sie doppelt genaue Variablen verwenden.

Permission Denied
Fehlernummer: 70
"Erlaubnis verweigert"

Dieser Fehler tritt meist dann auf, wenn die Diskette, auf die Sie schreiben wollen, schreibgeschützt ist. Wollen Sie wirklich darauf schreiben, entnehmen Sie die Diskette aus dem Laufwerk (Rote Lampe darf nicht leuchten!), schieben Sie den Schreibschutz-Schieber auf die andere Stellung und legen Sie die Diskette wieder ein.

Rename across disks
Fehlernummer: 74
"Umbennen über verschiedene Disketten"

Sie haben versucht, beim NAME-Befehl in einem Dateinamen einem neuen Diskettennamen anzugeben. Den Diskettennamen dürfen Sie aber nicht ändern, da das Resultat ja sonst ein Kopieren und kein Umbenennen wäre.

RESUME without error
Fehlernummer: 20
"RESUME, ohne daß ein Fehler aufgetreten ist."

AmigaBASIC hat einen RESUME-Befehl gefunden, es ist aber kein Error aufgetreten. Möglichweise hat sich das Programm in die ERROR-Schleife verirrt. Vor ERROR-Schleifen und Unterprogrammen sollten Sie immer das Hauptprogramm beenden

oder hinter den ON ERROR/RESUME-Block springen lassen. Wie Unterprogramme werden nämlich auch ON ERROR/RESUME-Blocks nicht automatisch übersprungen.

RETURN without GOSUB
Fehlernummer: 3
"RETURN ohne zugehöriges GOSUB"

AmigaBASIC hat einen RETURN-Befehl gefunden, ohne daß der aktuelle Programmteil mit GOSUB angesprungen wurde. Dieser Fehler passiert am häufigsten dadurch, daß ein Programm sich versehentlich in ein Unterprogramm verirrt. Bedenken Sie, daß im Gegensatz zu SUB-Programmen Unterprogramme nicht automatisch übersprungen werden. In der letzten Zeile vor dem Unterprogramm müssen Sie Ihr Programm an eine andere Stelle schicken.

SHARED outside of a subprogram
Fehlernummer: -
"SHARED-Befehl außerhalb eines SUB-Programms"

Der Befehl SHARED darf nur innerhalb eines SUB-Programms verwendet werden. Vielleicht ist das END SUB ein paar Zeilen zu weit nach oben gerutscht? Oder Sie haben EXIT SUB und END SUB verwechselt. Diesen Fehler teilt Ihnen AmigaBASIC vor der Programmausführung mit.

Statement illegal within subprogram
Fehlernummer: -
"Befehl innerhalb des SUB-Programms verboten"

Einige Befehle dürfen innerhalb von SUB-Programmen nicht verwendet werden. Im einzelnen sind verboten: DEF FN, der COMMON-Befehl und der CLEAR-Befehl. Auf diese Befehle müssen Sie zwischen SUB und END SUB verzichten. Amiga-BASIC merkt solche Fehler vor dem Programmstart.

String formula too complex
Fehlernummer: 16
"Die String-Formel ist zu komplex, zu verschachtelt"

Wenn Sie diesen Fehler erhalten, haben Sie eine Formel verwendet, die zuviele verschachtelte String-Befehle (MID$, LEFT$, RIGHT$ u.ä.) enthält. Sie sollten die Formel in mehrere Einzel-Formeln zerlegen und in verschiedenen Programmzeilen ausführen lassen.

String too long
Fehlernummer: 15
"Der String ist zu lang"

Ein String darf in AmigaBASIC 32767 Zeichen lang sein. Wenn er länger wird, gibt's diesen Fehler. Zerlegen Sie den String einfach in kleinere Teil-Strings.

SUB already defined
Fehlernummer: -
"SUB-Programm bereits definiert"

Kommt vor, wenn Sie zwei SUB-Programme verwenden, die den gleichen Namen haben. Eines der beiden müssen Sie umtaufen. Diesen Fehler erkennt AmigaBASIC vor der Programmausführung.

Subprogram already in use
Fehlernummer: 36
"SUB-Programm wird bereits benutzt"

Ein SUB-Programm darf zwar andere SUB-Programme aufrufen, aber nicht sich selbst. Steht also der eigene SUB-Aufruf innerhalb einer SUB-Routine, gibt's diesen Fehler. Diese Fehlermeldung kommt auch vor, wenn Sie ein SUB-Programm ab-

brechen und dann neu aufrufen. Beenden Sie in diesem Fall das alte SUB-Programm mit CONT oder starten Sie das Programm mit RUN neu.

Subscript out of range
Fehlernummer: 9
"Feldelement außerhalb des gültigen Bereichs"

Dieser Fehler kommt vor, wenn Sie ein Element eines Datenfelds benutzen, das außerhalb des Bereichs liegt, der mit DIM (oder automatisch) dimensoniert wurde. Oder wenn Sie mehr Dimensionen angeben, als durch DIM festgelegt wurde. Der Fehler ist typisch für einen Tippfehler im Feldnamen. Vielleicht verwenden Sie zur Angabe des Feldelements auch eine Variable, die im Programm einen falschen Wert erhalten hat.

SUB without END SUB
Fehlernummer: -
"SUB ohne zugehöriges END SUB"

Die Befehle SUB und END SUB gehören paarweise zueinander. Es gibt ein SUB, zu dem AmigaBASIC kein END SUB findet. Wie bei allen Befehlsklammern (FOR...NEXT, IF/END IF, WHILE...WEND etc.) hilft strukturiertes Programmieren, solche Fehler zu verhindern.

Syntax error
Fehlernummer: 2
"Syntax-Fehler, Satzbau-Fehler"

AmigaBASIC hat zwar Teile des eingegebenen Befehls verstanden, aber andere Teile sind falsch geschrieben, stehen in einer falschen Reihenfolge oder in falschem Zusammenhang. Vielleicht paßt auch die Anzahl offener und geschlossener Klammern nicht zusammen. 'Syntax error' tritt auch auf, wenn in ei-

ner DATA-Zeile ein String steht, der mit READ in eine normale Variable eingelesen werden soll. Checken Sie das alles ab. Schauen Sie auch mal in den Anhang B, wenn Sie nicht sicher sind, wie ein bestimmter Befehl verwendet wird.

Too many files
Fehlernummer: 67
"Zu viele Dateien"

Dieser Fehler bedeutet bei den anderen BASIC-Versionen von Microsoft, daß zu viele Dateien auf Diskette stehen. Beim Amiga kann das nicht passieren, da die Disketten ganz anders organisiert sind als bei herkömmlichen Computern. Ist kein Platz mehr, bekommen Sie ja einen "Disk full"-Error. Die Anzahl an erlaubten offenen Dateien ist in AmigaBASIC auf 256 beschränkt. So viele Dateien werden Sie aber fast nie gleichzeitig benutzen. Dieser Fehler sollte also eigentlich zu den seltenen Besuchern auf Ihrem Bildschirm gehören.

Tried to declare SUB within a SUB
Fehlernummer: -
"Sie versuchten, ein SUB-Programm innerhalb eines SUB-Programms zu definieren"

Innerhalb eines SUB-Programms darf kein anderes SUB-Programm stehen, das ist verboten. Sie können aber die SUB-Programme hintereinander schreiben und sich gegenseitig aufrufen lassen. Dieser Fehler wird vor der Programmausführung festgestellt.

Type mismatch
Fehlernummer: 13
"Typ stimmt nicht überein"

Dieser Fehler tritt immer dann auf, wenn bei der Übergabe von Werten oder der Zuweisung von Variablen verschiedene Zahlen-Darstellungen ohne Umwandlung ineinander überführt werden

sollen. Lesen Sie mal das Zwischenspiel 7, dann verstehen Sie, worum's geht. Auch ein SWAP-Befehl darf nur Variablen des gleichen Typs benutzen.

Undefined array
Fehlernummer: 38
"Unbekanntes Datenfeld"

Hinter SHARED haben Sie den Namen eines Felds angegeben, der im Programm nicht existiert. Vielleicht meinten Sie gar kein Datenfeld? Dann müssen Sie die Klammern hinter dem Variablennamen löschen.

Undefined label
Fehlernummer: 8
"Unbekanntes Label, unbekannte Sprungmarke"

Sie verwenden ein Label (z.B. hinter GOTO oder GOSUB), das es im Programm nicht gibt. Schauen Sie mal nach Tippfehlern im Namen. Gern vergißt man auch den Doppelpunkt hinter Labels. (Ist uns auch schon passiert.) Der muß aber hinter einem Label stehen, wenn das Label eine Zeile oder einen Programmteil benennt.

Undefined subprogram
Fehlernummer: 35
"Unbekanntes SUB-Programm"

Kommt dann vor, wenn Sie ein SUB-Programm aufrufen, das es nicht gibt. Oft wissen Sie gar nicht, daß Sie versucht haben, ein SUB-Programm zu aktivieren: Wenn AmigaBASIC einen Befehl aufgrund eines Tippfehlers nicht erkennt, sucht es nach einem SUB-Programm gleichen Namens. Wird auch dabei nichts gefunden, erscheint der "Undefined subprogram"-Error. Diese Meldung ist deshalb eine der häufigsten Fehlermeldungen.

Undefined user function
Fehlernummer: 18
"Unbekannte Benutzerfunktion"

Dieser Fehler tritt auf, wenn Sie eine Funktion FN... verwenden wollen, die nicht mit DEF FN definiert wurde. Vielleicht haben Sie sich beim Funktionsnamen vertippt? Oder ein Variablenname beginnt mit FN (darf er nämlich nicht, um Verwechslungen auszuschließen). Bedenken Sie auch, daß CLEAR sämtliche Funktionsdefinitionen mitlöscht.

Unknown volume
Fehlernummer: 49
"Unbekannter Diskettenname"

Dieser Fehler tritt auf, wenn Sie einen Diskettennamen angeben, der von AmigaBASIC nicht gefunden wird. Ursache kann ein Tippfehler sein. Wenn Sie einen Diskettennamen angeben, den AmigaBASIC nicht findet, fragt zunächst AmigaDOS durch ein Dialog-Window nach der Diskette. Klicken Sie in diesem Dialog-Window ins Feld "Cancel", wird AmigaBASIC mitgeteilt, daß die Diskette nicht gefunden werden konnte, wodurch die genannte Fehlermeldung entsteht.

Unprintable error
Fehlernummer: 21, 24, 25 ,27, 28, 31, 32, 39-48, 56, 59, 60, 65, 69, 71, 72, 75, 76, 78-255
"Nicht druckbarer Fehler"

Ein Fehler ist aufgetreten, für den AmigaBASIC keinen Fehlertext besitzt. Das kommt hauptsächlich dann vor, wenn Sie mit ON ERROR und dem ERROR-Befehl eigene Fehler-Routinen schreiben. Wenn Sie dabei eigene Fehlernummern definieren, müssen diese von ON ERROR natürlich auch abgefragt werden.

WEND without WHILE
Fehlernummer: 30
"WEND ohne zugehöriges WHILE"

Zu jedem WEND-Befehl muß auch ein WHILE gehören. Tritt dieser Fehler auf, hat AmigaBASIC mindestens ein überzähliges WEND gefunden. Wenn Sie Ihre Programme in strukturierter Schreibweise schreiben, ist die Gefahr geringer, ein WEND zu vergessen. Da WHILE und WEND immer paarweise aufeinander bezogen werden, suchen Sie einfach zu jedem WHILE im Programm das WEND, so finden Sie den Fehler.

WHILE without WEND
Fehlernummer: 29
"WHILE ohne zugehöriges WEND"

Dasselbe wie 'WEND without WHILE', nur umgekehrt: AmigaBASIC hat mindestens ein überzähliges WHILE gefunden. Vergleichen Sie 'WEND without WHILE'.

Anhang B: BASIC-Referenzteil

Dieser Anhang hat's nun wirklich in sich: Hier finden Sie alle, aber auch wirklich alle AmigaBASIC-Befehle aufgeführt und erklärt.

Wir haben die Befehle alphabetisch geordnet. Sie finden deshalb neben den Befehlen, die Sie schon aus dem vorderen Teil kennen, auch einige unbekannte Befehle. Bei diesen unbekannten Befehlen müssen Sie unter Umständen etwas mehr experimentieren als im vorderen Teil, da es oft viele Optionen gibt, die wir zwar vollständig aufführen, aber aus Platzgründen nicht so ausführlich wie gewohnt erklären können.

In dieser Reihenfolge finden Sie die Informationen zu den einzelnen Befehlen:

- Befehl. Hier steht der Name des Befehls, ohne Anhängsel und Optionen.
- Syntax. In dieser Zeile stehen alle Optionen des Befehls. Alles, was in eckigen Klammern steht, können Sie weglassen. Hat der Befehl keine weiteren Optionen, steht in dieser Zeile nichts.
- Erklärung. Nun folgt eine genaue Erklärung mit Hinweisen und gegebenenfalls Beispielen.
- Kapitel. Hier steht die Nummer des Kapitels, wo der Befehl vorne im Text erklärt wird.

ABS

Syntax: `Wert=ABS(Wert)`

Die Funktion ABS liefert den Absolutwert einer Zahl. Der Absolutwert ist der vorzeichenfreie Wert einer Zahl. ABS(-2) ist also 2, ABS(0) ist 0.

Kapitel: 1.17

AND

Syntax: **Wert = Wert1 AND Wert2**

Die logische Verknüpfung AND verknüpft die einzelnen Bits von 'Wert1' und 'Wert2' folgendermaßen:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Kapitel: Zwischenspiel 4

AREA

Syntax: **AREA [STEP] (x,y)**

Mit AREA geben Sie einen Punkt eines Vielecks an. Der AREAFILL-Befehl zeichnet dieses Vieleck auf den Bildschirm. Näheres finden Sie dort. Geben Sie hinter AREA die Option STEP an, werden die Koordinatenwerte 'x' und 'y' zur letzten verwendeten Koordinate dazugezählt (bzw. bei negativen Werten abgezogen).

Kapitel: 2.6

AREAFILL

Syntax: **AREAFILL [Wert für Darstellungsmodus]**

Bis zu zwanzig Eckpunkte können Sie mit AREA angeben. Mehr kann sich AmigaBASIC nicht merken. Kommt nun im Programm der Befehl AREAFILL, wird das Vieleck, das durch die AREA-Befehle angegeben wurde, auf einen Schlag auf den Bildschirm gebracht. Dank Blitter geht das unglaublich schnell.

Sie können einen von zwei Darstellungsmodi angeben: 0 bewirkt, daß das Vieleck in dem Muster ausgefüllt wird, den Sie mit PATTERN angegeben haben. 1 bewirkt, daß das Vieleck invertiert wird. Bei den normalen Farben heißt das:

Aus Blau wird Orange.
Aus Weiß wird Schwarz.
Aus Schwarz wird Weiß.
Und aus Orange wird Blau.

Kapitel: 2.6

ASC

Syntax: `Wert=ASC(String)`

ASC liefert den ASCII-Code des ersten Zeichens des in Klammern angegebenen Strings. Ist der String leer, gibt es einen "Illegal function call"-Error. Eine Tabelle aller ASCII-Codes finden Sie bei CHR$.

Kapitel: 4.3

ATN

Syntax: `Wert=ATN(Wert)`

ATN liefert den Arcustangens des Werts in Klammern. Sie erhalten einen Winkel im Bogenmaß. Er liegt zwischen -½*Pi und +½*Pi. Wollen Sie einen solchen Winkel ins Gradmaß umrechnen, brauchen Sie folgende Formel:

Grad = (180 * Bogenmaß)/Pi

oder

Grad = 57.296 * Bogenmaß

BEEP

Syntax:

Dieser Befehl erzeugt einen Piepser und ein kurzes Aufblitzen auf dem Bildschirm. Damit kann der Programmbenutzer auf etwas Besonderes hingewiesen werden. Den Piepser hört man nur, wenn mindestens der linke Tonausgang (Kanal 0) mit einem Lautsprecher verbunden ist. Anstelle von BEEP können Sie auch PRINT CHR$(7) verwenden, das funktioniert genauso.

Kapitel: 1.6

BREAK ON
BREAK OFF
BREAK STOP

Syntax: -

Diese Befehle steuern Event Trapping für einen Programmabbruch. Sie können nämlich auch das Drücken von <CTRL>-<C> oder das Wählen von "Stop" aus dem "Run"-Pulldown zu einem überprüfbaren Ereignis machen. So können Sie verhindern, daß ein Anwender Ihr Programm abbricht, oder darauf hinweisen, daß das zur Zeit nicht möglich ist. Mit ON BREAK GOSUB legen Sie das zuständige Unterprogramm fest. BREAK ON aktiviert Event Trapping, BREAK OFF deaktiviert es, und BREAK STOP schaltet es vorübergehend aus.

CALL

Syntax: **[CALL] Label [(Wert, ...)]**
[CALL] Variable [(Wert, ...)]

Zum einen rufen Sie mit dem CALL-Befehl SUB-Programme auf. Näheres über diese Programme erfahren Sie bei SUB...STATIC. CALL können Sie in den meisten Fällen weglas-

sen. Nur wenn es Verwechslungen geben könnte (z.B. mit normalen Unterprogramm-Labels), ist CALL zwingend.

Mit dem CALL-Befehl können Sie zum anderen auch Unterprogramme in Maschinensprache aufrufen. Sie wissen ja, daß der Amiga nicht von Haus aus BASIC spricht, sondern einen Übersetzer (den *Interpreter*) braucht. Wenn Sie sich mit dem Amiga in seiner Muttersprache unterhalten wollen, müssen Sie 68000 Assembler können. Ist das nicht der Fall, können Sie die nächsten Absätze gern überspringen.

Sie müssen das Maschinenspracheprogramm zunächst irgendwie in den Speicher bekommen. Dazu können Sie es von Diskette laden oder mit READ aus DATA-Feldern lesen. Die Startadresse der Maschinenroutine erfahren Sie mit den Funktionen VARPTR oder SADD. Ein Beispiel:

```
DIM Maschinenprogramm%(20)
FOR x=0 TO 20
  READ Maschinenprogramm%(x)
NEXT x
StartAdresse&=VARPTR(Maschinenprogramm%(0))
CALL StartAdresse&(10,20)
DATA .......
```

Das ganze Programm steht also im Integerfeld 'Maschinenprogramm%'. Die Adresse, ab der dieses Feld im Speicher steht, erfahren Sie mit

```
VARPTR(Maschinenprogramm%(0)).
```

Das ist nämlich die Startadresse des ersten Feldelements. Da sie theoretisch bis zu 24 Bits lang sein kann, verwenden wir eine 32-Bit-Integerzahl (mit einem & am Ende). Mit dem CALL-Befehl können Sie dem Maschinenspracheprogramm auch noch Parameter übergeben, wie in unserem Beispiel die Zahlen 10 und 20. Auch in einem String kann das Maschinenspracheprogramm abgelegt werden:

```
Ma$=""
FOR x=0 TO 20
  READ Wert
  Ma$=Ma$+MKI$(Wert)
NEXT x
StartAdresse&=SADD(Ma$)
CALL StartAdresse&(10,20)
DATA .......
```

Die Funktion SADD liefert die Anfangsadresse eines Strings. Mit MKI$ setzen wir den String aus den Einzelwerten zusammen, die in der DATA-Zeile stehen müssen. Drittens können Sie mit CALL auch Maschinen-Unterprogramme aus einer Library (einer "Bibliothek") aufrufen. Näheres dazu beim Befehl LIBRARY.

Kapitel: 4.5

CDBL
CINT
CLNG
CSNG

Syntax: `Wert=C***(Wert)`

Diese Befehle wandeln einen Zahlenwert in ein anderes Darstellungsformat um.

CDBL erzeugt eine doppelt genaue Fließkommazahl.

CINT erzeugt eine 16-Bit-Integerzahl. Dabei wird die Zahl nach den üblichen Regeln gerundet: CINT(3.4)=3, CINT(3.5)=4. Der Wert in Klammern darf zwischen -32768 und 32767 liegen.

CLNG erzeugt eine 32-Bit-Integerzahl. Auch hier wird gerundet. Der Wert in Klammern darf zwischen -2147483648 und 2147483647 liegen.

CSNG erzeugt eine einfach genaue Fließkommazahl.

Ist die Ursprungszahl für den gewünschten Zahlentyp zu groß, gibt's einen "Overflow"-Error. Zu den verschiedenen Zahlentypen lesen Sie bitte Zwischenspiel 7.

Kapitel: 1.17

CHAIN

Syntax: **CHAIN [MERGE] Dateiname [,Zeile] [,ALL] [,DELETE Zeile-Zeile oder Label-Label]**

Dieser Befehl ermöglicht es einem BASIC-Programm, ein anderes BASIC-Programm nachzuladen und ihm die Kontrolle zu übergeben. Der Unterschied zum MERGE-Befehl ist, daß hier das aufrufende Programm ganz oder teilweise gelöscht wird. Beim CHAIN-Befehl ist das Wort MERGE eine von mehreren Optionen. Wenn Sie sie verwenden, wird das aufrufende Programm ab der 'Zeile', die Sie hinter dem Dateinamen angegeben haben, von dem neuen Programm überschrieben. Das nachgeladene Programm muß dazu als ASCII-Datei auf Diskette stehen.

Fehlt MERGE, gibt 'Zeile' die Zeilennummer im nachgeladenen Programm an, ab der das Programm gestartet werden soll. Für 'Zeile' darf leider kein Label angegeben werden, Sie müssen in diesem Fall Zeilennummern verwenden. Das ist etwas inkonsequent von Microsoft, aber in der aktuellen Version von AmigaBASIC leider nicht zu ändern. Die Option ,ALL bewirkt, daß alle Variablen des aktuellen Programms dem nachgeladenen Programm übergeben werden. Wollen Sie das nicht, können Sie mit dem Befehl COMMON einzelne Variablen auswählen, die übergeben werden sollen. Wenn Sie die Variablenübergabe mit dem COMMON-Befehl lösen wollen, darf ,ALL nicht angegeben werden.

,DELETE löscht im nachgeladenen Programm einen bestimmten Zeilenbereich. Sie können ihn durch Zeilennummern oder Labels angeben.

Der CHAIN-Befehl setzt den Zeiger für DATA-Elemente auf den Anfang zurück, wirkt also wie RESTORE. Neben den Variablen bleiben nach CHAIN auch alle geöffneten Dateien offen und Einstellungen wie durch OPTION BASE erhalten. Andererseits schaltet CHAIN das Event Trapping aus. Sie müssen es im neuen Programm reaktivieren. Auch Typ-Festlegungen mit DEFINT, DEFLNG etc. werden vergessen. Steht eine Festlegung mit dem DEF FN-Befehl nicht mehr im neu erzeugten Programm, vergißt AmigaBASIC auch die alten Funktions-Definitionen.

CHDIR

Syntax: **CHDIR " [Gerät oder Laufwerk oder Diskettenname:] [Inhaltsverzeichnis] [/Inhaltsverzeichnis ...]**

CHDIR ändert das aktuelle Inhaltsverzeichnis. Sie können dabei entweder vom aktuellen Inhaltsverzeichnis aus "tiefer" in die Unter-Directories steigen oder von ganz vorn einen neuen Weg durch die Verzweigungen angeben. Wie ein System von Unter-Inhaltsverzeichnissen aufgebaut ist, zeigt Ihnen z.B. Bild 13 in Kapitel 3.2.

Mit CHDIR "/" steigen Sie in den Inhaltsverzeichnissen eine Ebene nach oben. Wollen Sie mit diesem Befehl von der höchsten Stufe ("Basis-Verzeichnis") eine weitere Ebene nach oben steigen, erhalten Sie einen "File not found"-Error. Die gleiche Meldung kommt natürlich auch, wenn Sie ein Unter-Inhaltsverzeichnis angeben, das nicht existiert. Ins Basis-Verzeichnis gelangen Sie mit CHDIR ":".

Kapitel: 3.2

CHR$

Syntax: String=CHR$(Wert)

Die Funktion CHR$ erzeugt das Zeichen, das zu dem in Klammern angegebenen ASCII-Code gehört. Der Wert in Klammern darf zwischen 0 und 255 liegen. Der ASCII-Code (American Standard Code for Information Interchange) ist ein genormter Code, der jedem druckbaren Zeichen und jedem Steuerzeichen eine Zahl zwischen 0 und 255 zuordnet. Die Zeichen 128 bis 255 sind bisher noch nicht vereinheitlicht, der Amiga benutzt sie für Funktionstasten und landesspezifische Sonderzeichen. Die Zeichen, die ein eckiges Kästchen ergeben, sind noch frei. Bild 24 zeigt eine Tabelle der ASCII-Codes.

Kapitel: 1.15

0	[CTRL]-[@]		32		64	@	96	`
1	[CTRL]-[A]		33	!	65	A	97	a
2	[CTRL]-[B]		34	"	66	B	98	b
3	[CTRL]-[C]	(Break)	35	#	67	C	99	c
4	[CTRL]-[D]		36	$	68	D	100	d
5	[CTRL]-[E]		37	%	69	E	101	e
6	[CTRL]-[F]		38	&	70	F	102	f
7	[CTRL]-[G]	(Beep)	39	'	71	G	103	g
8	[CTRL]-[H]	<BACKSPACE>	40	(	72	H	104	h
9	[CTRL]-[I]	<TAB>	41	)	73	I	105	i
10	[CTRL]-[J]	(Line feed)	42	*	74	J	106	j
11	[CTRL]-[K]		43	+	75	K	107	k
12	[CTRL]-[L]	(Löschen)	44	,	76	L	108	l
13	[CTRL]-[M]	<RETURN>	45	-	77	M	109	m
14	[CTRL]-[N]		46	.	78	N	110	n
15	[CTRL]-[O]		47	/	79	O	111	o
16	[CTRL]-[P]		48	0	80	P	112	p
17	[CTRL]-[Q]		49	1	81	Q	113	q
18	[CTRL]-[R]		50	2	82	R	114	r
19	[CTRL]-[S]		51	3	83	S	115	s
20	[CTRL]-[T]		52	4	84	T	116	t
21	[CTRL]-[U]		53	5	85	U	117	u
22	[CTRL]-[V]		54	6	86	V	118	v
23	[CTRL]-[W]		55	7	87	W	119	w
24	[CTRL]-[X]		56	8	88	X	120	x
25	[CTRL]-[Y]		57	9	89	Y	121	y
26	[CTRL]-[Z]		58	:	90	Z	122	z
27	[CTRL]-[[]	<ESC>	59	;	91	[	123	{
28	[CTRL]-[\]	<hoch>	60	<	92	\	124	|
29	[CTRL]-[]]	<runter>	61	=	93	]	125	}
30	[CTRL]-[^]	<rechts>	62	>	94	^	126	~
31	[CTRL]-[_]	<links>	63	?	95	_	127	[illegible]

Bild 24a: Die ASCII-Code-Tabelle (Teil 1)

Code	Zeichen	Taste	Code	Zeichen	Code	Zeichen	Code	Zeichen
128	□		160		192	À	224	à
129	□	⟨F1⟩	161	¡	193	Á	225	á
130	□	⟨F2⟩	162	¢	194	Â	226	â
131	□	⟨F3⟩	163	£	195	Ã	227	ã
132	□	⟨F4⟩	164	¤	196	Ä	228	ä
133	□	⟨F5⟩	165	¥	197	Å	229	å
134	□	⟨F6⟩	166	¦	198	Æ	230	æ
135	□	⟨F7⟩	167	§	199	Ç	231	ç
136	□	⟨F8⟩	168	¨	200	È	232	è
137	□	⟨F9⟩	169	©	201	É	233	é
138	□	⟨F10⟩	170	ª	202	Ê	234	ê
139	□	⟨HELP⟩	171	«	203	Ë	235	ë
140	□		172	¬	204	Ì	236	ì
141	□		173	-	205	Í	237	í
142	□		174	®	206	Î	238	î
143	□		175	¯	207	Ï	239	ï
144	□		176	°	208	Ð	240	ð
145	□		177	±	209	Ñ	241	ñ
146	□		178	²	210	Ò	242	ò
147	□		179	³	211	Ó	243	ó
148	□		180	´	212	Ô	244	ô
149	□		181	µ	213	Õ	245	õ
150	□		182	¶	214	Ö	246	ö
151	□		183	·	215	□	247	□
152	□		184		216	Ø	248	ø
153	□		185	¹	217	Ù	249	ù
154	□		186	º	218	Ú	250	ú
155	□		187	»	219	Û	251	û
156	□		188	¼	220	Ü	252	ü
157	□		189	½	221	Ý	253	ý
158	□		190	¾	222	Þ	254	þ
159	□		191	¿	223	ß	255	ÿ

Bild 24b: Die ASCII-Code-Tabelle (Teil 2)

CINT

siehe CDBL

CIRCLE

Syntax: **CIRCLE [STEP] (x,y),Radius [,Farbe] [,Anfangswinkel] [,Endwinkel] [,x/y-Verhältnis]**

Dieser Befehl zeichnet einen Kreis oder genaugenommen eine Ellipse um den Mittelpunkt (x,y). 'Radius' gibt den x-Radius in Pixels an. Wenn Sie STEP vor dem Mittelpunkt verwenden, zählt AmigaBASIC die Werte 'x' und 'y' zur letzten verwendeten Grafik-Koordinate dazu.

'Farbe' gibt die Farbnummer an, in der gezeichnet werden soll. Die Anzahl der erlaubten Farben hängt von der Anzahl der Bitebenen im verwendeten Screen ab. 'Anfangswinkel' und 'Endwinkel' ermöglichen das Zeichnen von Kreisausschnitten: Geben Sie einen Bogenmaß-Winkel (Vielfaches von Pi) für den Anfangspunkt und einen für den Endpunkt an. Negative Winkel bewirken, daß der Anfangs- und der Endpunkt mit dem Mittelpunkt verbunden werden. Bild 6 in Kapitel 2.6 hilft Ihnen beim Bestimmen der Winkel.

'x/y-Verhältnis' legt das Verhältnis des x-Radius zum y-Radius fest. So können Sie alle Arten von Ellipsen erzeugen. Werte unter 0.44 bewirken einen kleinen y-Radius, eine liegende Ellipse entsteht. Über 0.44 wird der y-Radius größer als der x-Radius, eine stehende Ellipse wird gezeichnet. Der Wert 0.44 entspricht normalerweise (abhängig von der Monitoreinstellung) einem perfekt runden Kreis.

Kapitel: 2.6

CLEAR

Syntax: CLEAR [,BASIC-Speicherbereich] [,Stack]

Zunächst dient CLEAR dazu, alle Variablen, Strings und Datenfelder zu löschen. Auch alle offenen Dateien werden durch diesen Befehl geschlossen. Sie können aber noch ein bißchen mehr damit anfangen.

CLEAR ermöglicht es auch, die Speichereinteilung von AmigaBASIC zu verändern. Sie können den Speicherbereich für das BASIC-Programm und seine Daten nach Bedarf vergrößern oder verkleinern. Die Mindestgröße für den 'BASIC-Speicherbereich' ist 1024 Bytes, also 1 KByte. Nach oben ist die einzige Grenze die Größe des Speicherplatzes im System. In Grafikprogrammen braucht der Amiga viel Speicher, um die Grafiken darzustellen. Lassen Sie also bitte genug Speicherplatz übrig. Wenn dem Amiga Systemspeicherplatz fehlt, kommt es gern zum Systemabsturz ("Guru Meditation" - Sie kennen das ja).

Bei Programmen, die hauptsächlich mit Daten arbeiten und nicht viel Grafikspeicher brauchen, können Sie den BASIC-Speicher aber ziemlich weit nach oben setzen. Die Voreinstellung für den BASIC-Speicherbereich ist 25000 Bytes auf dem 512K-Amiga.

Ein Beispiel:

```
CLEAR ,40000
```

stellt 40000 Bytes für den BASIC-Speicher zur Verfügung.

Der 'Stack' ist der Speicherbereich, den AmigaBASIC für interne Zwischenwerte etc. benutzt. Hier merkt es sich z.B. den Inhalt von Zählvariablen bei FOR...NEXT oder die Zeile, in die das Programm nach RETURN zurückkehren soll. Dieser Bereich ist auf 4789 Bytes voreingestellt. Seine Mindestgröße ist 1024 Bytes. Sehen Sie sich auch den Befehl FRE(x) an. Da steht, wie Sie die aktuellen Speichergrößen feststellen können.

CLNG

siehe CDBL

CLOSE

Syntax: **CLOSE [[#] Dateinummer] [, [#] Dateinummer]**

Mit CLOSE schließen Sie eine oder mehrere Dateien. Das Schließen von Dateien hat mehrere Gründe:

Erstens legt AmigaBASIC für jede Datei einen Pufferspeicher an. Erst durch den CLOSE-Befehl wird beim Schreiben der letzte Pufferinhalt wirklich übertragen. Zweitens muß AmigaDOS beim Schließen von Ausgabedateien auf Diskette einige Informationen auf den neuesten Stand bringen (z.B. die Dateilänge).

Und drittens wird die verwendete Dateinummer dadurch wieder für neue OPEN-Befehle freigegeben. Und bei OPEN finden Sie auch Näheres über Dateien.

Kapitel: 3.3

CLS

Syntax: -

CLS löscht den Bildschirminhalt des aktuellen Ausgabewindows. Der Ausgabecursor wird danach in die linke obere Ecke gesetzt. Das bedeutet, die nächste Ausgabe mit PRINT oder einem ähnlichen Befehl erscheint an dieser Position.

Kapitel: 1.3

COLLISION

Syntax: Wert=COLLISION(Objektnummer)

Die COLLISION-Funktion liefert verschiedene Ergebnisse, die Auskunft über Kollisionen zwischen Bobs oder Sprites geben. Jede Kollision wird in eine Warteschlange geschrieben, so daß alle Zusammenstöße abgefragt und bearbeitet werden können. Allerdings kann sich AmigaBASIC nicht mehr als 16 Kollisionen gleichzeitig merken.

In Klammern geben Sie die Nummer des Objekts an, das auf einen Zusammenstoß untersucht werden soll. Das Ergebnis ist dann die Nummer des Objekts, mit dem das angegebene Objekt kollidierte. Die Kollision wird aus der Warteschlange gelöscht. Es ist auch möglich, daß die Kollision mit dem Window-Rahmen stattfand. Dann erhalten Sie ein Ergebnis zwischen -1 und -4:

-1	oberer Rand
-2	linker Rand
-3	unterer Rand
-4	rechter Rand

Anstelle einer 'Objektnummer' können Sie auch 0 oder -1 angeben: Die Funktion COLLISION(0) liefert die Nummer des Objekts, das zuletzt an einer Kollision beteiligt war, ohne die Kollision aus der Warteschlange zu löschen. Das ist dann nützlich, wenn Sie abfragen wollen, welche Kollision zuletzt stattfand, ohne sie dabei gleich zu bearbeiten.

Und COLLISION(-1) hat als Ergebnis die Nummer des Windows, in dem die Kollision erfolgte. Auch diese Information kann in Programmen wichtig sein, z.B. wenn mehrere Bewegungen in verschiedenen Windows stattfinden.

COLLISION ON
COLLISION OFF
COLLISION STOP

Syntax:

Diese drei Befehle sind für Event Trapping aufgrund von Objekt-Kollisionen zuständig. COLLISION ON schaltet die Kollisionsüberwachung ein, COLLISION OFF schaltet sie aus, COLLISION STOP unterbricht die Überwachung bis zum nächsten COLLISION ON. Mit ON COLLISION GOSUB können Sie das Unterprogramm angeben, zu dem das Programm bei einer Kollision verzweigen soll. Dort können Sie dann mit der COLLISION-Funktion die beteiligten Objekte untersuchen.

COLOR

Syntax: COLOR [Vordergrundfarbe] [,Hintergrundfarbe]

COLOR legt für jede Art von Ausgabe (Text, Grafik) eine 'Vordergrundfarbe' (= Textfarbe, Zeichenfarbe) und eine 'Hintergrundfarbe' (Bildschirmhintergrund) fest. Für beide Farben geben Sie jeweils eine Farbnummer an. Die Höhe der Farbnummer hängt von der Anzahl der Bit-Ebenen im verwendeten Screen ab. Welchen Farben die Farbnummern entsprechen, wird durch die Einstellungen in Preferences und mit dem PALETTE-Befehl bestimmt. Wenn Sie einen Wert oder beide Werte weglassen, verwendet AmigaBASIC automatisch Farbe Nummer 1 (normalerweise weiß) als 'Vordergrundfarbe' und Nummer 0 (normalerweise blau) als 'Hintergrundfarbe'. Wenn Sie unmittelbar vor Textausgaben den Wert für die 'Hintergrundfarbe' ändern, erscheint der Text auf einem farbigen Balken. Bei den meisten Grafikbefehlen hingegen wird der Hintergrund gar nicht verändert.

Kapitel: 1.16

COMMON

Syntax: COMMON Variablenname [,Variablenname, ...]

Dieser Befehl übergibt einzelne, angegebene Variablen einem Programm, das Sie mit CHAIN nachladen. Wenn Sie Felder übergeben wollen, müssen Sie sie durch ein Paar leere Klammern kennzeichnen:

```
CHAIN "(Programmname)"
COMMON a,b$,Hallo%, Farben(), Texte$()
```

Sie dürfen auch mehrere COMMON-Anweisungen verwenden, jede Variable darf aber nur einmal vorkommen.

CONT

Syntax: -

Mit CONT setzen Sie ein Programm fort, das durch <CTRL>-<C>, "Stop" aus dem "Run"-Pulldown, einen STOP- oder einen END-Befehl im Programm abgebrochen wurde. Das Programm darf während der Unterbrechung aber nicht verändert werden, sonst erhalten Sie einen "Can't continue"-Error.

COS

Syntax: Wert=COS(Wert)

COS liefert den Cosinus eines Werts. Der Winkel muß im Bogenmaß angegeben werden. Wollen Sie einen Winkel im Gradmaß ins Bogenmaß umrechnen, brauchen Sie folgende Formel:

Bogenmaß = (Pi * Grad)/180

oder

Bogenmaß = 0.0175 * Grad

Kapitel: 2.5

CSNG

siehe CDBL

CSRLIN

Syntax: `Wert=CSRLIN`

CSRLIN ist eine sogenannte Systemvariable. Ähnlich wie DATE$ oder FRE(0) ist das eine Variable, die von AmigaBASIC einen bestimmten Wert zugewiesen bekommt. Der Anwender kann die Variable nicht verwenden. CSRLIN beinhaltet die Zeile, in der der Cursor zur Zeit steht. So können Berechnungen für LOCATE durchgeführt werden. Vergleichen Sie bitte auch POS.

CVD
CVI
CVL
CVS

Syntax:

```
Wert=CVD(8-Byte-String)
Wert=CVI(2-Byte-String)
Wert=CVL(4-Byte-String)
Wert=CVS(4-Byte-String)
```

Wenn Sie Zahlen in Dateien speichern, ist es platzsparend und schneller, sie als Bytes abzulegen, die als Strings gelesen und geschrieben werden. Dazu gibt es die MK*$-Funktionen. Um

aus solchen Strings beim Lesen wieder Zahlen zu machen, dienen diese vier Funktionen:

CVD macht aus einem 8 Byte langen String eine doppelt genaue Fließkommazahl.

CVI macht aus einem 2 Byte langen String eine 16-Bit-Integerzahl.

CVL macht aus einem 4 Byte langen String eine 32-Bit-Integerzahl.

CVS macht aus einem 4 Byte langen String eine einfach genaue Fließkommazahl.

Zu den Zahlentypen lesen Sie bitte Zwischenspiel 7.

Kapitel: 4.1, 4.3

DATA

Syntax: **DATA Wert [,Wert, ...]**

In DATA-Zeilen können Sie Werte (Zahlen oder Strings) ablegen, die mit dem READ-Befehl gelesen werden. Die Werte werden durch Kommas getrennt. Enthält ein String Kommas oder Strichpunkte, muß er in Anführungszeichen geschrieben werden.

DATA-Zeilen dürfen sich an jeder Stelle im Programm befinden, die DATA-Werte werden einzeln nacheinander gelesen.

Kapitel: 2.9

DATE$

Syntax: String=DATE$

Diese Funktion liefert das Systemdatum in einem 10 Zeichen langen String. Das "Systemdatum" ist das Datum, das in Preferences eingestellt ist bzw. das Datum, das AmigaDOS aus einer eingebauten Echtzeituhr (A500 und A2000) übernommen hat. Das Datum wird im amerikanischen Format MM-TT-JJJJ dargestellt. In Kapitel 4.7 finden Sie eine Routine, die DATE$ ins deutsche Format umrechnet.

Kapitel: 1.17, 4.7

DECLARE FUNCTION ... LIBRARY

Syntax: DECLARE FUNCTION Name [Variablenname, ...] LIBRARY

Wenn Sie Maschinenprogramme aus einer Library verwenden (vergleichen Sie bitte CALL und LIBRARY), und diese Programme eine Funktion sind, also ein Ergebnis liefern, müssen Sie die Funktion mit DECLARE FUNKTION ... LIBRARY ankündigen oder anmelden. Soll das Ergebnis einen bestimmten Zahlentyp haben, hängen Sie einfach die Typ-Kennzeichnung an den Namen der Funktion an:

```
DECLARE FUNCTION Test%(Testwert) LIBRARY
```

Die Maschinensprache-Funktion 'Text%(Testwert)' liefert eine 16-Bit-Integerzahl als Ergebnis. Die Angabe der Variablen, die das Maschinenprogramm erwartet, ist nicht nötig, denn AmigaBASIC kümmert sich sowieso nicht darum. Wenn Sie sie trotzdem hinschreiben, wissen Sie aber auch noch später, welche Werte der Maschinensprache-Funktion eigentlich übergeben werden müssen.

DEF FN

Syntax: **DEF FNName [(Wert, [Wert,...])] = Definition**

Mit DEF FN können Sie eigene BASIC-Funktionen festlegen. Der Funktionsname wird hinter FN angehängt. Wenn Sie die Funktion im Programm verwenden wollen, schreiben Sie z.B. PRINT FNa(100) oder Test=FNText(1,2,4). Die Werte, die Sie optional in Klammern angeben, und die in der Funktionsdefinition vorkommen (DEF FNa(x) = 2*x), werden beim Funktionsaufruf durch den dort angegebenen Wert ersetzt. Das Ergebnis der Funktion ist das Ergebnis der Formel, die Sie in der Definition angegeben haben, für die übergebenen Werte ausgerechnet. Kommen in der Definition andere Variablen vor als hinter FNName angegeben, werden beim Ausrechnen die gerade aktuellen Werte dieser Variablen verwendet:

```
DEF FNTest(x) = 2*x*a
a=100 : PRINT FNTest(25)
a=12 : PRINT FNTest(25)
```

ergibt 5000 und 1250. Sie können auch mit Stringfunktionen eigene Funktionen definieren.

```
DEF FNErst$(a$) = LEFT$(a$,1)
PRINT FNErst$("Amiga")
```

ergibt ein A.

Kapitel: 7.6

DEFDBL
DEFINT
DEFLNG
DEFSNG
DEFSTR

Syntax: **DEF*** Buchstabe [-Buchstabe] [,Buchstabe...]**

Mit diesen Befehlen können Sie bestimmte Variablennamen auf einen bestimmten Variablentyp festlegen. Dazu geben Sie den Anfangsbuchstaben der Variablennamen an:

DEFINT a-c bedeutet, daß ab sofort alle Variablen, die mit den Buchstaben a bis c beginnen (also Amiga, alpha, Beta, Charlie,...), 16-Bit-Integervariablen sein sollen. Sie müssen nun nicht mehr jede einzelne Variable mit % kennzeichnen. Das gilt auch für Datenfelder. Wenn Sie aber eine Variable aus dem festgelegten Bereich als anderen Typ kennzeichnen (Antwort$), geht diese Kennzeichnung vor. 'Antwort$' ist also trotzdem ein String, wogegen 'Antwort' als 16-Bit-Integervariable verarbeitet wird. Folgende Festlegungen sind möglich:

Befehl	Variablentyp	Beispielvariable für den Variablentyp
DEFINT	16-Bit-Integervariable (engl.: "Short INTeger")	Hallo%
DEFLNG	32-Bit-Integervariable ("LoNG Integer")	Hallo&
DEFSNG	einfach genaue Fließkomma-var. ("SiNGle Precision")	Hallo, Hallo!
DEFDBL	doppelt genaue Fließkomma-var. ("DouBLe Precision")	Hallo#
DEFSTR	String ("STRing")	Hallo$

Die Festlegung des Variablentyps muß natürlich vor der Verwendung der Variablen erfolgen.

DELETE

Syntax: DELETE [Label oder Zeilennummer] [-] [Label oder Zeilennummer]

Mit DELETE können Sie sowohl im Direktmodus als auch im Programm Zeilenbereiche löschen. Geben Sie einfach an, von wo bis wo Sie löschen wollen:

```
DELETE Vorbereitungen - Farbdefinition
```

löscht alle Zeilen zwischen den beiden Labels.

```
DELETE Vorbereitungen -
```

löscht alle Zeilen ab 'Vorbereitungen:' bis zum Programmende.

```
DELETE - Farbdefinition
```

löscht alle Zeilen vom Programmanfang bis zum Label 'Farbdefinition:'.

DIM

Syntax: DIM [SHARED] Feldname(Wert [,Wert,...]) [,Feldname (...),...]

Mit DIM dimensionieren Sie Datenfelder, die Sie im Programm verwenden wollen. Fehlt diese DIMensionierung, legt Amiga-BASIC das Feld automatisch auf 10 Elemente fest. Das Ganze dient dazu, für ein Feld möglichst nur so viel Speicherplatz zu verwenden wie wirklich nötig. Geben Sie hinter DIM das Wort SHARED an, sind die angegebenen Datenfelder auch in allen SUB-Programmen verwendbar. Der DIM SHARED-Befehl darf aber nur im Hauptprogramm verwendet werden. Wenn Sie eine Reihe normaler Variablen ebenfalls in allen SUB-Programmen verwenden wollen, können Sie dazu auch den DIM SHARED-Befehl einsetzen:

```
DIM SHARED Farben%(31,2), Maxfarbe
```

Die Variable 'Maxfarbe' ist nun eine sogenannte "globale" Variable, im Gegensatz zu den lokalen Variablen in den SUB-Programmen.

Mit dem Befehl OPTION BASE können Sie festlegen, ob das kleinste Feldelement aller Datenfelder 0 oder 1 ist. Sie können für ein Feld mehrere Dimensionen angeben (DIM Feld(2,2,2)), höchstens jedoch 255. Das höchste von AmigaBASIC erlaubte Feldelement hat die Nummer 32767. Die beiden Grenzwerte für Dimensionen und Feldelemente werden Sie aber wahrscheinlich nie erreichen, weil vorher der Speicherplatz zu Ende gehen dürfte.

Kapitel: 1.7

ELSE

siehe IF...THEN...ELSE

END

Syntax: -

Dieser Befehl beendet ein BASIC-Programm. Dabei werden alle offenen Dateien geschlossen.

Kapitel: 2.10

END SUB

Syntax: END SUB

Beendet ein SUB-Programm. Bitte lesen Sie die Erklärungen zum SUB-Befehl.

Kapitel: 4.5

EOF

Syntax: **Wert=EOF(Dateinummer)**

Die EOF-Funktion stellt fest, ob in einer Datei, die gelesen wird, noch weitere Daten stehen. EOF(Dateinummer) liefert das Ergebnis 0, wenn noch Daten zum Lesen anstehen, und -1, wenn die Datei zuende ist. Durch diese Funktion kann Ihr Programm feststellen, ob es noch Daten lesen darf oder nicht. Wollen Sie nämlich Daten einlesen, die es gar nicht gibt, erhalten Sie die Fehlermeldung "Input past end".

Kapitel: 3.3

EQV

Syntax: **Wert = Wert1 EQV Wert2**

Die logische Verknüpfung EQV verknüpft die einzelnen Bits von 'Wert1' und 'Wert2' folgendermaßen:

0 EQV 0 = 1
0 EQV 1 = 0
1 EQV 0 = 0
1 EQV 1 = 1

Kapitel: Zwischenspiel 4

ERASE

Syntax: **ERASE Feldname [,Feldname...]**

ERASE löscht ein Datenfeld. Der benötigte Speicherplatz wird wieder freigegeben, die Feldinhalte werden gelöscht. Nach ERASE können Sie ein Feld neu mit DIM dimensionieren.

Kapitel: 4.8

ERL

Syntax: **Wert=ERL**

Diese Systemvariable liefert bei Error-Trapping (Event Trapping für Fehler) die letzte vor dem Fehler verwendete Zeilennummer. Leider gibt es keine Möglichkeit, den Namen des Labels zu erfahren, das vor dem Error steht. Weiteres finden Sie bei ON ERROR GOSUB.

ERR

Syntax: **Wert=ERR**

Neben ERL noch eine Systemvariable für Error Trapping. Diesmal erfahren Sie die Fehlernummer des Errors, der die Unterbrechung bzw. das Ereignis (Event) verursacht hat. Welche Fehlernummer zu welchem Fehler gehört, erfahren Sie im Anhang A. Und weiteres zu dem Thema steht bei ON ERROR GOSUB.

ERROR

Syntax: **ERROR Fehlernummer**

Ein dritter Befehl, bei dem es um Errors geht. Durch Error Trapping mit ON ERROR GOSUB können Sie in Ihrem Programm eigene Fehlermeldungen produzieren. Sie nehmen einfach eine Fehlernummer, die noch frei ist, und erzeugen mit dem ERROR-Befehl diesen Fehler. Error Trapping verzweigt zum zuständigen Unterprogramm, wo Sie aus der Variablen ERR Ihre Fehlernummer auslesen können. Es ist nicht möglich, im Error-Window von AmigaBASIC eigene Fehlermeldungen auszugeben. Sie können aber natürlich in Ihrem Unterprogramm ein eigenes Error-Window erzeugen.

Wenn Sie eine Fehlernummer angeben, die von AmigaBASIC verwendet wird, und kein Error Trapping benutzen, erzeugen Sie einen Fehler, der eigentlich gar nicht aufgetreten ist. Das könnten Sie zum Beispiel einsetzen, wenn Sie Ihr Programm gegen unbefugte Benutzer schützen wollen. Versuchen Sie mal:

```
ERROR 2
```

Ein "Syntax error" wird erzeugt.

EXIT SUB

Syntax: -

Dieser Befehl ermöglicht das vorzeitige Beenden eines SUB-Programms. Näheres finden Sie bei SUB...STATIC.

Kapitel: 4.8

EXP

Syntax: **Wert=EXP(Wert)**

EXP errechnet das Ergebnis der Exponentialfunktion (für Mathematiker: e^x, für Nicht-Mathematiker: Vergessen Sie's bitte gleich wieder!).

FIELD

Syntax: **FIELD [#] Dateinummer, Länge AS String [,Länge AS STRING,...]**

Diesen Befehl benutzen Sie für relative Dateiverwaltung. Er dient dazu, einen Datensatzpuffer einzurichten und verschiedene Stringvariablen als Übergabevariablen für den Puffer zu definieren. 'Länge' ist die Anzahl an Zeichen im jeweiligen Feld, 'String' ist ein Variablenname. Wollen Sie im Datensatzpuffer 10

Bytes für den Vornamen und 20 Bytes für den Nachnamen anlegen, geht das so:

```
FIELD #1, 10 AS Vorname, 20 AS Nachname
```

Diesen Variablen darf das Programm nur mit den Befehlen LSET oder RSET neue Werte zuweisen, sonst wird die Zuordnung zwischen Variable und Puffer aufgehoben. Die Gesamtlänge aller festgelegten Felder darf nicht länger sein als die Satzlänge, die Sie beim OPEN-Befehl für die Datei angeben. Sonst erhalten Sie einen "Field overflow"-Error. Wenn Sie für dieselbe Datei eine neue FIELD-Zeile angeben, wird der Puffer vom ersten Byte an neu eingeteilt.

Kapitel: 5.1

FILES

Syntax: FILES [Inhaltsverzeichnis]

FILES zeigt das aktuelle Inhaltsverzeichnis auf dem Bildschirm an. Das aktuelle Inhaltsverzeichnis können Sie mit CHDIR ändern. Wenn Sie hinter FILES ein Inhaltsverzeichnis angeben, können Sie sich dessen Inhalt anzeigen lassen, ohne es extra zum aktuellen Inhaltsverzeichnis zu machen. Das ist besonders zum Durchsuchen verschiedener Inhaltsverzeichnisse praktisch.

Kapitel: 3.2

FIX

Syntax: Wert=FIX(Wert)

Diese Funktion schneidet den Nachkommateil einer Zahl ab. FIX(3.235325) ist 3, FIX(-2.3532325) ist -2. Im Gegensatz zu den Funktionen INT und CINT wird nicht gerundet.

FOR...NEXT

Syntax: **FOR Variablenname=Anfangswert TO Endwert [STEP Schrittweite]**
NEXT [Wert] [,Wert...]

Dieser Befehl ermöglicht Zählschleifen. Das heißt, alle Befehle, die zwischen FOR und NEXT stehen, werden mehrfach ausgeführt. Beim ersten Schleifendurchlauf hat die Zählvariable den Anfangswert, beim nächsten Durchlauf den Anfangswert +1. So geht es weiter, bis die Zählvariable den Endwert erreicht hat. Mit STEP können Sie optional auch noch angeben, in welchen Schritten gezählt werden soll. Beim NEXT-Befehl können Sie die Zählvariable angeben oder auch weglassen. Es ist auch möglich, mehrere FOR...NEXT-Schleifen mit einem einzigen NEXT-Befehl zu schließen:

```
FOR x=1 TO 100
  FOR y=1 TO 20
    FOR z=1 TO 30
NEXT z,y,x
```

Wie Sie in dem Beispiel sehen, können Sie FOR...NEXT-Schleifen auch verschachteln. Die inneren Schleifen müssen vor den äußeren geschlossen werden.

Kapitel: 1.7

FRE(x)

Syntax: -

Diese Systemvariable gibt Auskunft über die Größe der einzelnen Speicherbereiche. Welchen Speicherbereich Sie abfragen wollen, geben Sie hinter FRE in Klammern an: FRE(0) liefert den freien Platz im BASIC-Speicherbereich. Das ist die Anzahl an Bytes, die zur Zeit vom Programm und seinen Variablen nicht benötigt werden.

FRE(-1) liefert den freien Platz im Systemspeicher. FRE(-2) liefert die Anzahl der Bytes im Stack, die noch nicht von AmigaBASIC benötigt wurden.

Kapitel: 4.4

GET (für Grafik)

Syntax: **GET (x1,y1)-(x2,y2), Feldname [(Feldposition,...)]**

Mit diesem Befehl können Sie den Inhalt eines Bildschirmbereichs als Ausschnitt in einem Datenfeld abspeichern. Das Feld 'Feldname' muß ein Zahlenfeld sein. Zur Auswahl stehen Integerfelder, Fließkommafelder einfacher und Fließkommafelder doppelter Genauigkeit. Folgende Formel errechnet die benötigte Größe des Felds in Byte:

6 + Bitebenen * Höhe * 2 * INT((Breite + 16) / 16)

Das Ergebnis können Sie dann durch die Anzahl an Bytes pro Feldelement teilen.

2 Bytes bei einem Integerfeld (Hallo%(x))
4 Bytes bei einem einfach genauen Fließkommafeld (Hallo(x))
8 Bytes bei einem doppelt genauen Fließkommafeld (Hallo#(x))

Im Feldelement 0 steht die Breite, im Element 1 die Höhe und in Element 2 die Anzahl der Bitebenen des gespeicherten Bildausschnitts.

Wenn Sie ein mehrdimensionales Feld verwenden, können Sie durch Ändern der Feldpositionen mehrere Ausschnitte in einem einzigen Datenfeld abspeichern. Oder auch verschiedene An-

sichten derselben Grafik, die dann schnell gewechselt werden können.

Um den gespeicherten Ausschnitt wieder auf dem Bildschirm zu bringen, benutzen Sie den PUT-Befehl.

Kapitel: 4.1

GET (Daten)

Syntax: **GET [#] Dateinummer [,Satznummer]**

GET liest bei relativer Dateiverwaltung den angegebenen Datensatz in den Dateipuffer ein. Wenn Sie die 'Satznummer' nicht angeben, wird automatisch die nächste Satznummer verwendet. Die 'Satznummer' muß theoretisch zwischen 0 und 16777215 liegen. Praktisch ist die Speicherkapazität des verwendeten Geräts (Floppy, Festplatte, RAM-Disk) für die höchstmögliche Nummer verantwortlich. Sie liegt auf jeden Fall deutlich unter 16777215.

Nach dem GET-Befehl können Sie den Pufferinhalt auslesen. Das geht entweder über die im FIELD-Befehl definierten Übergabevariablen oder auch mit den Befehlen INPUT# und LINE INPUT#.

Kapitel: 5.1

GOSUB...RETURN

Syntax: **GOSUB Label**
RETURN [Label]

Mit GOSUB rufen Sie ein Unterprogramm auf. AmigaBASIC merkt sich, wo das GOSUB steht und verzweigt zum angegebenen Label. Findet es in diesem Unterprogramm einen RE-

TURN-Befehl, kehrt das Programm zu dem Befehl zurück, der nach dem Aufruf durch GOSUB steht, und macht dort weiter.

Sie können auch hinter RETURN ein Label angeben. Dann kehrt AmigaBASIC nicht zum Befehl hinter GOSUB zurück, sondern zum angegebenen Programmteil. Dabei müssen Sie jedoch aufpassen, daß der GOSUB-Aufruf nicht innerhalb einer anderen Klammer bzw. Schleife wie FOR...NEXT, WHILE...WEND oder in einem anderen Unterprogramm stand. Sonst werden diese Klammern, Schleifen oder Unterprogramme unter Umständen nicht richtig oder gar nicht beendet, was zu Fehlermeldungen wie "FOR without NEXT" oder "WHILE without WEND" führen kann.

Kapitel: 1.16

GOTO

Syntax: `GOTO Label`

Dieser Befehl bewirkt, daß das Programm zum angegebenen Label oder zu einer angegebenen Zeilennummer springt und dort weiterarbeitet.

Kapitel: 1.6

HEX$

Syntax: `String=HEX$(Wert)`

Diese Funktion liefert die hexadezimale Schreibweise einer Zahl zwischen -32768 und 65535.

HEX$(60037) ergibt EA85.

Mehr über Hexadezimalzahlen erfahren Sie in Zwischenspiel 4.

IF...THEN...ELSE

Syntax:
```
IF (Bedingung) THEN (Befehle) [ELSE (Befehle)]
IF (Bedingung) GOTO Label [ELSE (Befehle)]

IF (Bedingung) THEN
  (Befehle)
[ELSE IF (Bedingung) THEN
  (Befehle)]
[ELSE
  (Befehle)]
END IF
```

Der Befehl IF...THEN ermöglicht es, Bedingungen zu prüfen und je nach Ergebnis bestimmte Befehle oder Verzweigungen auszuführen.

Sie sehen, es gibt sehr verschiedene Syntax-Versionen. Der einfachste IF...THEN-Befehl sieht so aus:

IF (Bedingung) THEN (Befehl)

Die Bedingung ist immer ein Vergleich, der ein logisches Ergebnis hat (z.B. IF a<10... oder IF Hallo=0...). Trifft die Bedingung zu, vergibt AmigaBASIC das Ergebnis -1 (wahr). Trifft sie nicht zu, ist das Ergebnis 0 (falsch). Die oben gezeigte Variante führt die Befehle hinter THEN aus, falls die Bedingung zutrifft. Sie können diesen Befehl um ELSE erweitern und dahinter angeben, was das Programm tun soll, wenn die Bedingung nicht zutrifft:

```
IF a<10 THEN PRINT "a kleiner als 10" ELSE PRINT "a nicht kleiner
als 10"
```

Anstelle des Befehls hinter THEN können Sie auch GOTO verwenden und zu einem Label oder einer Zeilennummer verzweigen lassen.

Sollen abhängig von einer Bedingung mehrere Programmzeilen ausgeführt werden, empfiehlt sich die IF/ELSE IF/ELSE/END IF-Struktur: Die Zeile IF...THEN legt die Bedingung fest. Hinter

THEN darf kein Befehl mehr stehen. Die Befehle, die unter dieser Zeile bis zum nächsten END IF, ELSE oder ELSE IF stehen, werden ausgeführt, wenn die Bedingung zutrifft. Die optionale Zeile ELSE IF...THEN gibt eine zweite Bedingung an, die nur geprüft werden soll, wenn die erste Bedingung nicht zutrifft. Auch hier darf hinter THEN nichts mehr stehen. Die Befehle, die bei zutreffender zweiter Bedingung ausgeführt werden sollen, beginnen eine Zeile später.

Die optionale Zeile ELSE gibt an, daß die folgenden Befehle ausgeführt werden sollen, wenn die zuletzt überprüfte Bedingung (IF...THEN oder ELSE IF...THEN) nicht zutrifft. END IF beendet die ganze IF/ELSE IF/ELSE-Struktur. Solche Strukturen können Sie auch ineinander verschachteln, z.B.:

```
IF ... THEN
  IF ... THEN
   ...
  ELSE
   ...
  END IF
ELSE IF ... THEN
  IF ... THEN
   ...
  END IF
ELSE
   ...
END IF
```

Strukturierte Programmierung ist bei so etwas natürlich schon fast unverzichtbar.

Kapitel: 1.6, 2.9

IMP

Syntax: **Wert = Wert1 IMP Wert2**

Die logische Verknüpfung IMP verknüpft die einzelnen Bits von 'Wert1' und 'Wert2' folgendermaßen:

0 IMP 0 = 1
0 IMP 1 = 1
1 IMP 0 = 0
1 IMP 1 = 1

Kapitel: Zwischenspiel 4

INKEY$

Syntax: **Wert$=INKEY$**

Dieser Befehl liest ein Zeichen aus dem Tastaturpuffer und übergibt es als String mit der Länge 1 Zeichen. Wurde keine Taste gedrückt, ist der Tastaturpuffer also leer, liefert INKEY$ einen leeren String. INKEY$ zeigt selbst kein Zeichen auf dem Bildschirm an, es übergibt das Zeichen nur.

Kapitel: 1.13

INPUT

Syntax: **INPUT [;] ["(Hinweistext),"] [;] Variable, [Variable, Variable, ...]**

Mit INPUT können Sie den Benutzer Werte auf der Tastatur eingeben lassen, die dann der angegebenen Variablen übergeben werden. Wenn Sie wollen, kann der INPUT-Befehl vorher einen Hinweistext auf den Bildschirm drucken. Folgt dem Text ein Strichpunkt (;), gibt AmigaBASIC hinter dem Text ein Fragezeichen aus. Geben Sie hinter dem Text ein Komma an (,), wird kein Fragezeichen gedruckt. Sie können mehrere Variablenwerte eingeben lassen, dann müssen bei der Eingabe die Einzelwerte

durch Kommas getrennt werden. Stimmen der eingegebene Wert und die zugewiesene Variable nicht überein oder gibt der Anwender zu wenig oder zu viele Daten ein, erscheint die Fehlermeldung '?Redo from start'. Das bedeutet, die Eingabe muß von vorne wiederholt werden. Wenn Sie bem Programmieren vor dem Hinweistext einen Strichpunkt angeben, steht der Cursor nach der Abarbeitung des INPUT-Befehls unmittelbar hinter dem letzten eingegebenen Zeichen. Dort erscheint dann die nächste Ausgabe.

Kapitel: 1.6

INPUT$

Syntax: **String=INPUT$(Länge [, [#] Dateinummer])**

Dieser Befehl wird bei sequentiellen Dateien benutzt. Er liest einen String mit einer angegebenen Länge aus der angegebenen Datei. Wenn Sie die Dateinummer weglassen, liest INPUT$ die gewünschte Anzahl an Zeichen von der Tastatur ein. Bei diesem Einlesen erscheint zwar der Textcursor, aber Ihre Tastatureingaben werden während des Eintippens trotzdem nicht auf dem Bildschirm dargestellt. Das Einlesen von der Tastatur kann nicht mit der <RETURN>-Taste abgebrochen werden. Nur <CTRL>-<C> ist ein Ausweg. Wenn Sie mit INPUT$ Bytes einlesen, können Sie diese mit den CV*-Funktionen in Zahlen zurückverwandeln.

Kapitel: 4.1

INPUT#

Syntax: **INPUT# Dateinummer, Variable [,Variable,...]**

Dieser Befehl wirkt wie der normale Bildschirm-INPUT, nur liest er Variablen oder Strings aus einer angegebenen Datei. Diese Datei muß natürlich vorher mit OPEN eröffnet worden

sein. Die Daten, die INPUT# einliest, wurden vorher mit PRINT# oder WRITE# in die Datei geschrieben.

INPUT# liest eine Variable bis zum nächsten Trennzeichen in der Datei. Als Trennzeichen erkennt dieser Befehl Leerzeichen, Kommas, Carriage-Return-Codes (CHR$(10)) und Zeilenvorschub-Codes (CHR$(13)). Bei Strings gilt das Leerzeichen nicht als Trennzeichen. Wollen Sie Strings einlesen, die Kommas beinhalten, müssen die Strings in Anführungszeichen in die Datei geschrieben worden sein.

Kapitel: 3.3

INSTR

Syntax: **Wert=INSTR([Wert,] String,Suchstring)**

Diese Funktion sucht den angegebenen Suchstring im angegebenen String. Wenn sie den Suchstring im String findet, liefert sie die Position, ab der der Suchstring im String steht. Wird der String nicht gefunden, hat INSTR das Ergebnis 0. Ebenso ist das Ergebnis 0, wenn der Suchstring größer als der durchsuchte String oder leer ist.

Optional können Sie vor dem durchsuchten String die Position angeben, ab der gesucht werden soll. Ist diese Position größer als die Länge des durchsuchten Strings, ist das Ergebnis wieder 0.

INSTR("Amiga","iga") ergibt 3.
INSTR("Hallo","iga") ergibt 0.
INSTR(1,"Test","es") ergibt 2.
INSTR(3,"Test","es") ergibt 0.

INT

Syntax: **Wert=INT(Wert)**

Diese Funktion erzeugt aus einer Fließkommazahl eine ganze Zahl. Zu den Funktionen CINT und FIX besteht folgender Unterschied: Bei positiven Werten werden einfach die Nachkommastellen abgeschnitten, bei negativen Werten wird abgerundet. INT liefert also die ganze Zahl, die kleiner oder gleich dem angegebenen Wert ist.

INT(3.4) ist 3.
INT(3.8) ist auch 3.
INT(-2.2) ist -3.

Außerdem gibt es bei INT keine Einschränkung für den erlaubten Zahlenbereich.

Kapitel: 1.17

KILL

Syntax: **KILL Dateiname**

Mit KILL können Sie Dateien auf Diskette, Festplatte oder aus der RAM-Disk löschen. Dateien, die zur Zeit geöffnet sind, können nicht gelöscht werden.

Kapitel: 3.2

LBOUND
UBOUND

Syntax: **Wert=LBOUND(Feldname [, Dimension])**
Wert=UBOUND(Feldname [, Dimension])

Diese Funktion liefert die Nummer des kleinsten (Lower BOUNDary = untere Grenze) und des größten Feldelements (Upper BOUNDary = obere Grenze) eines Datenfelds. So können Sie die Angaben überprüfen, die mit DIM und OPTION BASE gemacht wurden.

Ein Beispiel:

```
OPTION BASE 0 : DIM a(200)
```

LBOUND(a) liefert 0, UBOUND(a) liefert 200.

Wenn Sie zusätzlich einen Wert für die 'Dimension' angeben, können Sie die Unter- und Obergrenze einzelner Dimensionen herausfinden:

```
DIM a(3,4,56)
```

UBOUND(a,1) liefert die Größe der ersten Felddimension, also 3. UBOUND(a,2) liefert 4 und UBOUND(a,3) liefert 56. Wenn Sie den Wert 'Dimension' weglassen, erhalten Sie die Ausmaße der ersten Felddimension.

LEFT$

Syntax: **String=LEFT$(String, Anzahl)**

LEFT$ bildet einen String, der den linken Teil des angegebenen Strings beinhaltet. 'Anzahl' ist die Anzahl der Zeichen im erzeugten Teilstring. Da ein String in AmigaBASIC nicht größer

als 32767 Zeichen sein darf, muß 'Anzahl' zwischen 0 und 32767 liegen.

LEFT$("Amiga ist toll!",5) ergibt "Amiga".

Kapitel: 1.16

LEN

Syntax: **Wert=LEN(String)**

LEN ergibt die Länge eines Strings in Zeichen. Eventuelle Leerzeichen und Steuerzeichen im String werden mitgezählt:

LEN("Amiga ist toll!") ergibt 15.

Kapitel: 1.17, 2.7

LET

Syntax: **LET Variablenname=Wert**

Der 'Wert' wird der angegebenen Variablen zugewiesen. Den Befehl LET können Sie auch weglassen, es genügt eine Zuweisung wie: a=10.

Kapitel: 1.3

LIBRARY

Syntax: **LIBRARY "(Dateiname)"**

Wenn Sie in Ihrem Programm Maschinensprache-Unterprogramme verwenden wollen, müssen Sie die nicht unbedingt selbst schreiben. Sie können auch sogenannte Libraries verwenden ("Bibliotheken"). In diesen Libraries stehen eine ganze Reihe von Maschinenprogrammaufrufen aus dem Betriebssystem, die

bestimmte nützliche Funktionen erfüllen. Sie benötigen allerdings Informationen über die Parameter und Parametertypen, die von so einer Routine aus einer Library erwartet werden. Dabei kann Ihnen eigentlich nur die Dokumentation helfen, die es für Maschinensprache-Programmierer und Software-Entwickler gibt: Das ROM-KERNAL-Handbuch und das Intuition-Handbuch. Dort sind nämlich alle Betriebssystemroutinen erklärt, die Sie über Libraries verwenden können.

Ohne diese Informationen läuft nichts. Deshalb haben wir in diesem Buch auch auf ein Beispiel zu LIBRARY verzichtet. Das Programm "Library" in der "BASICDemos"-Schublade auf der Extras-Diskette zeigt Interessierten, wie's geht. Die Routinen werden mit CALL aufgerufen. Im SUB-Programm 'Font' aus dem "Library"-Demoprogramm steht z.B. CALL CloseFont(pFont&). Die Routine 'CloseFont' ist Bestandteil der Library "graphics.library", die zu Beginn des Programms angesprochen wird. Solche Libraries müssen im .BMAP-Format vorliegen. Das bedeutet, daß AmigaBASIC wenn Sie z.B. die Library "Dos.library" aufrufen, eine Datei namens "Dos.bmap" benötigt. Solche .BMAP-Dateien finden Sie zum Teil bereits auf der "ExtrasD"-Diskette. Zum Teil müssen Sie sie auch erst mit dem Programm "ConvertFD" erzeugen.
Lesen Sie zu diesem Thema bitte auch die Erklärungen zu den Programmen "Library" und "ConvertFD" im Anhang C. AmigaBASIC kann gleichzeitig bis zu fünf Libraries benutzen.

Neben den Programmaufrufen mit CALL können Sie auch Maschinenprogramm-Funktionen verwenden. Solche Funktionen (im Gegensatz zu den CALL-Routinen liefern sie ein Ergebnis) müssen mit dem Befehl DECLARE FUNCTION ... LIBRARY angekündigt werden. Im "Library"-Demo gibt es z.B. die Funktionen 'AskSoftStyle&','OpenFont&' und 'Execute&'.

LINE

Syntax: **LINE [[STEP] (x1,y1)] - [STEP] (x2,y2) [,Farbe] [,B oder ,BF]**

Dieser Befehl wird verwendet, um eine Linie, ein umrandetes Rechteck oder ein ausgemaltes Rechteck zu zeichnen. Sie können einfach einen Anfangs- und einen Endpunkt angeben (LINE (x1,y1)-(x2,y2)), dann entsteht eine Linie. Das ist die einfachste Möglichkeit.

Wenn Sie den ersten Punkt weglassen (LINE -(x2,y2)), zieht AmigaBASIC eine Linie vom letzten verwendeten Grafikpunkt zum angegebenen Punkt. Diese beiden Möglichkeiten können Sie auch in Verbindung mit STEP einsetzen. Dann werden die angegebenen x- und y-Werte zum letzten Grafikpunkt dazugezählt. Optional ist auch eine Farbnummer, mit der Sie die Zeichenfarbe bestimmen. Ganz am Schluß können Sie ',B' anhängen, dann wird ein Rechteck gezeichnet. Die Koordinaten (x1,y1) und (x2,y2) geben zwei diagonal gegenüberliegende Ecken des Rechtecks an. Und ',BF' bewirkt dasselbe, allerdings wird das Rechteck mit der Zeichenfarbe gefüllt.

Kapitel: 2.5

LINE INPUT

Syntax: **LINE INPUT [;] ["(Hinweistext);"] [;] String-Variable**

Der LINE INPUT-Befehl liest eine String-Variable von der Tastatur ein. Sie können beliebige Zeichen eingeben (also auch Satzzeichen wie Kommas, Strichpunkte etc.), die alle in den angegebenen String übernommen werden. Nur die <RETURN>-Taste beendet die Eingabe. Sie können wieder einen Hinweistext angeben, aber im Gegensatz zu INPUT nur eine einzige Variable. Diese Variable muß eine String-Variable sein, sonst erhalten Sie einen 'Type mismatch'-Error. LINE INPUT erzeugt grundsätzlich kein Fragezeichen hinter dem Hinweistext - egal,

ob Sie zwischen Text und Variable ein Komma oder einen Strichpunkt angeben. Ein Strichpunkt hinter dem Befehl LINE INPUT und vor dem Hinweistext bewirkt (genau wie bei INPUT), daß die nachfolgende Bildschirmausgabe direkt an die Benutzereingabe angehängt wird.

Kapitel: 1.7

LINE INPUT#

Syntax: **LINE INPUT # Dateinummer, Variable**

Dieser Befehl wirkt genauso wie LINE INPUT, allerdings liest er die Zeichen aus einer Datei. LINE INPUT erkennt nur CHR$(10) und CHR$(13) als Trennzeichen an. Alle anderen Zeichen (also auch Leerstellen und Kommas) werden in den String übernommen.

LIST

Syntax: **LIST [Zeilennummer oder Label] [- Zeilennummer oder Label] [,Datei]**

Mit diesem Befehl können Sie Ihr Programm auflisten lassen. Dabei gibt es die Möglichkeit, von Anfang an (LIST) oder von einer bestimmten Zeile an zu listen (LIST Vorbereitungen -). Sie können eine weitere Zeilennummer angeben, dann wird bis zu dieser Zeile gelistet. (LIST - Farbdef; LIST Muster-Farbdef) Normalerweise werden Programme im LIST-Window gelistet. Hinter LIST können Sie einen Dateinamen oder einen Gerätnamen angeben. Mit LIST, "SCRN:" listen Sie z.B. ein Programm im BASIC-Window, mit LIST, "PRT:" auf dem Drucker. Wenn Sie ein Programm in eine Diskettendatei listen lassen (LIST, "DF0:Test"), wird das Programm im ASCII-Format abgespeichert.

Kapitel: 1.5

LLIST

Syntax: **LLIST [Zeilennummer oder Label] [-] [Zeilennummer oder Label]**

LLIST funktioniert wie LIST, schickt das Listing aber auf den Drucker, der am Amiga angeschlossen ist. Den Druckertyp und die Schnittstelle, die er benutzt, können Sie in Preferences einstellen.

Weiteres finden Sie bei LIST.

Kapitel: 3.5

LOAD

Syntax: **LOAD [Dateiname] [,R]**

LOAD lädt ein Programm von Diskette, Festplatte oder aus der RAM-Disk. Wenn Sie hinter dem Namen des Programms ein ,R angeben, wird das Programm direkt gestartet. Wenn Sie keinen Programmnamen angeben, bittet Sie AmigaBASIC in einem Dialog-Window, den Namen einzugeben.

Kapitel: 1.13

LOC

Syntax: **Wert=LOC(Dateinummer)**

Die LOC-Funktion liefert die Nummer des Datenblocks auf Diskette, Festplatte oder RAM-Disk, der zuletzt gelesen oder geschrieben wurde. Obwohl dieser Wert nur von der Einteilung eines Speichermediums durch AmigaDOS abhängt, gibt er bei relativen Dateien die Nummer des zuletzt gelesenen Datensatzes an. Das liegt daran, daß bei relativen Dateien ein übertragener

Block im Puffer immer genau einem Datensatz entspricht. Bei sequentiellen Dateien wird's schwieriger: Hier gibt LOC die Blocknummer abhängig von der Puffergröße an. Die Standardeinstellung für den Puffer ist 128 Bytes, Sie können sie aber im OPEN-Befehl verändern.

LOCATE

Syntax: LOCATE [Zeile] [,Spalte]

Dieser Befehl dient dazu, den Ausgabecursor an eine bestimmte Bildschirmposition zu bringen. Für 'Zeile' und 'Spalte' müssen Sie positive Werte angeben, also Werte größer als 0. Auch 0 selbst ist nicht erlaubt, die Bildschirmposition in der linken oberen Ecke ist (1,1). Bei 60-Zeichen-Einstellung ist 21 der höchste noch sichtbare Wert für die 'Zeile' und 62 der für die 'Spalte'. Bei 80-Zeichen-Einstellung ist 23 der höchste Zeilenwert und 77 der höchste Spaltenwert.

Kapitel: 1.7

LOF

Syntax: Wert=LOF(Dateinummer)

LOF gibt die Gesamtlänge einer Datei in Bytes an. Wenn Ihr Programm diesen Wert kennt, kann es beim Einlesen mitzählen oder (z.B. bei OBJECT.SHAPE) die ganze Datei auf einmal einlesen.

Kapitel: 4.1

LOG

Syntax: `Wert=LOG(Wert)`

LOG ergibt den natürlichen Logarithmus einer Zahl. (Für Mathematiker: Den Logarithmus zur Basis e. Für Teil-Mathematiker: e ist 2.718281828. Für Nicht-Mathematiker: Was soll's.) Der Logarithmus einer Zahl a zu einer Basis b ist die Zahl c, mit der man b potenzieren muß, um a zu erhalten: $b^c=a$. LOG(a) zur Basis b ist c.

Aus verschiedenen Gründen haben sich die Mathematiker besonders in die Zahl e als Basis verliebt. (2.71828... Na, Sie wissen ja schon.)

Wenn man schon mal den Logarithmus für eine Berechnung in BASIC braucht (z.B. in unserer PICSAVE-Routine), dann garantiert nicht zur Basis e. Deshalb hier die Formel zur Umrechnung auf andere Basen:

LOG(a) zur Basis b = LOG(a)/LOG(b)

Kapitel: 4.8

LPOS

Syntax: `Wert=LPOS(x)`

Wirkt wie POS, gibt aber die Position des letzten ausgegebenen Zeichens im Druckerpuffer an. So wissen Sie, wieviele Zeichen in der aktuellen Zeile schon an den Drucker geschickt wurden. Der Wert 'x' ist ein Scheinargument (Vergleichen Sie bitte POS).

LPRINT

Syntax: **LPRINT [Variable oder Wert] [Trennzeichen] [Variable oder Wert ...]**

Wirkt wie PRINT, schickt aber die Ausgaben auf den Drucker. Die Zeichen werden gepuffert und vom Drucker erst dann gedruckt, wenn ein Carriage Return-Code folgt (Carriage Return, CHR$(10)). Geben Sie hinter der letzen gedruckten Variablen kein Trennzeichen mehr an, schickt AmigaBASIC den CHR$(10) selbständig.

Weiteres bei PRINT.

Kapitel: 3.5

LPRINT USING

Syntax: **LPRINT USING (FormatString) ; (Variable oder Wert) [;]**

Funktioniert wie PRINT USING, die Ausgaben erscheinen aber auf dem Drucker. Weiteres bei LPRINT und PRINT USING.

LSET

Syntax: **LSET Übergabestring=String**

LSET wird bei relativer Dateiverwaltung benutzt. Er überträgt Daten in den Datensatz-Puffer. Dazu geben Sie eine Übergabevariable an, die Sie im FIELD-Befehl definiert haben, und weisen ihr einen String zu. Dieser String wird dann in den Puffer übertragen. Das L bei LSET kommt von "Linksbündig". Sind die Daten kürzer als die im FIELD-Befehl angegebene Länge des Felds, werden sie linksbündig in das Datenfeld übertragen. Das heißt, die freien Stellen befinden sich rechts. Vergleichen Sie

RSET. Zur Übernahme von Zahlenwerten in den Puffer müssen Sie Umwandlungsfunktionen wie MK*$ oder STR$ verwenden. Um die Daten vom Puffer auf Diskette zu bringen, brauchen Sie den PUT-Befehl.

Kapitel: 5.1

MENU

Syntax: **MENU Menünummer, Menüpunkt, Status [,Text]**

Mit MENU können Sie in AmigaBASIC eigene Pulldowns erzeugen. Durch Event Trapping ist es dann möglich, die Menüauswahl auszuwerten.

'Menünummer' gibt die Nummer des Pulldowns an. Das erste Pulldown hat die Nummer 1, das letzte Pulldown Nummer 10.

'Menüpunkt' ist der Punkt innerhalb des gewählten Pulldowns. In einem Pulldown dürfen bis zu 19 Menüpunkte stehen. Der Wert 0 legt den Titel des Pulldowns fest.

Für 'Status' sind drei Werte möglich: 0 schaltet den angegebenen Menüpunkt ab, er wird in Geisterschrift dargestellt und kann nicht ausgewählt werden. Wenn Sie als Menüpunkt 0 angeben, wird ein ganzes Pulldown abgeschaltet. 1 aktiviert den angegebenen Menüpunkt (oder bei Menüpunkt 0 das ganze Pulldown). 2 aktiviert ebenfalls, druckt aber noch ein kleines Häkchen zur Kennzeichnung von den Text des Menüpunkts. Sie sollten im Text zwei Leerstellen Platz lassen, damit das Häkchen nichts überschreibt. Im Menütitel kann kein Häkchen angegeben werden.

Wenn Sie den MENU-Befehl dazu verwenden, einen Menüpunkt zu aktivieren oder abzuschalten, war das alles. Wenn Sie aber ein neues Pulldown erzeugen, müssen Sie noch den Text angeben, den die einzelnen Menüpunkte haben sollen.

Kapitel: 2.8

MENU

Syntax: Wert=MENU(x)

Diese BASIC-Funktion gibt Auskunft über den letzten gewählten Menüpunkt. MENU(0) teilt Ihnen die Nummer des Pulldowns mit, in dem sich der Punkt befindet. MENU(1) sagt Ihnen, welcher Punkt innerhalb dieses Pulldowns ausgewählt wurde.

Kapitel: 2.8

MENU ON
MENU OFF
MENU STOP

Syntax: -

Diese Befehle aktivieren oder deaktivieren Event Trapping für die Menüsteuerung. MENU ON ermöglicht Menu Trapping, MENU OFF schaltet es ab, und MENU STOP unterdrückt es bis zum nächsten MENU ON.

Das Unterprogramm, das sich um die Menüauswahl kümmert, können Sie mit ON MENU GOSUB angeben.

MERGE

Syntax: MERGE Dateiname

Mit MERGE lesen Sie ein Programm oder ein Unterprogramm von Diskette und hängen es an das aktuelle Programm an. Das gelesene Programm muß im ASCII-Format auf Diskette stehen. Sie können MERGE zwar auch vom Programm ausführen lassen, aber nach Beendigung von MERGE kehrt AmigaBASIC immer in den Direktmodus zurück. Zum Verketten und Nachladen von Programmen ist CHAIN besser geeignet. MERGE hingegen ist ein Befehl, der eher während der Programmentwicklung benutzt wird.

Kapitel: Zwischenspiel 3

MID$

Syntax: Teilstring=MID$(String,Position [,Länge])

Mit der Funktion MID$ können Sie einen Teilstring aus einem String ermitteln. Geben Sie den String und die Position an, ab der der Teilstring gebildet werden soll. Außerdem die Länge des gewünschten Teilstrings. Wenn Sie diese Angabe weglassen, erhalten Sie den Rest des Strings ab der angegebenen Position.

MID$("Amiga ist toll!",11,4) ergibt "toll".

Kapitel: 4.7

MID$

Syntax: MID$(String,Position [,Länge])=Einsetz-String

MID$ gibt es auch als Befehl. Damit können Sie in einen vorhandenen String einen Einsetz-String einbauen. 'Position' gibt die Position an, ab der eingebaut werden soll. Optional können

Sie eine 'Länge' angeben. Dann werden nur so viele Zeichen aus dem Einsetz-String übernommen wie angegeben.

```
a$="Amiga ist nett!" : MID$(a$,11)="toll"
```

ergibt "Amiga ist toll!".

MKD$
MKI$
MKL$
MKS$

Syntax: **String=MKD$(doppelt genaue Fließkommazahl)**
String=MKI$(16-Bit-Integerzahl)
String=MKL$(32-Bit-Integerzahl)
String=MKS$(einfach genaue Fließkommazahl)

Wenn Sie Zahlen in sequentiellen oder relativen Dateien speichern, ist es platzsparend und schneller, sie als Strings zu speichern, die genauso viele Zeichen haben, wie die Zahl Bytes hat. Zur Umwandlung einer Zahl in so einen String gibt es diese Funktionen.

MKD$ macht aus einer doppelt genauen Fließkommazahl einen 8 Byte langen String.

MKI$ macht aus einer 16-Bit-Integerzahl einen 2 Byte langen String.

MKL$ macht aus einer 32-Bit-Integerzahl einen 4 Byte langen String.

MKS$ macht aus einer einfach genauen Fließkommazahl einen 4 Byte langen String.

Wie Sie solche Strings beim Einlesen wieder in Zahlen des entsprechenden Formats umwandeln können, lesen Sie bei den CV*-Befehlen.

Kapitel: 4.1, 4.3

MOUSE

Syntax: Wert=MOUSE(x)

Die MOUSE-Funktion gibt Informationen über die Klicks und Bewegungen, die der Anwender mit der Maus durchführt.

Der Wert in Klammern kann zwischen 0 und 6 liegen und hat folgende Bedeutung: MOUSE(0) gibt den Zustand der linken Maustaste an. Wird MOUSE(0) im Programm ausgeführt, merkt sich AmigaBASIC den gegenwärtigen Zustand der Maus. Bevor Sie MOUSE(1) bis (6) abfragen, sollten Sie also MOUSE(0) aufrufen. Wenn Sie den Wert nicht benötigen, weisen Sie ihn einfach einer Dummy-Variablen zu. MOUSE(0) kann folgende Ergebnisse haben:

Wert	Bedeutung
0	Die linke Maustaste ist zur Zeit nicht gedrückt.
1	Die linke Maustaste ist zur Zeit nicht gedrückt, der Anwender hat sie aber seit dem letzten Aufruf von MOUSE(0) einmal gedrückt.
2,3	Die linke Maustaste ist zur Zeit nicht gedrückt, sie wurde aber seit dem letzten MOUSE(0) mehrmals gedrückt. Der Wert gibt die Anzahl der Klicks an: 2 - zweimal gedrückt, 3 - dreimal oder öfter gedrückt.
-1	Die linke Maustaste wird zur Zeit gedrückt.
-2,-3	Die linke Maustaste wird zur Zeit gedrückt. Der aktuelle Klick ist der zweite bzw. dritte Klick seit dem letzten MOUSE(0).

MOUSE(1) und MOUSE(2) sagen Ihnen die x- bzw. y-Koordinate des Mauscursors beim letzten Aufruf von MOUSE(0).

MOUSE(3) und MOUSE(4) sagen Ihnen den Startpunkt der letzten Mausbewegung.

MOUSE(5) und MOUSE(6) teilen Ihnen den Endpunkt der letzten Mausbewegung mit. Ist die linke Maustaste zur Zeit noch gedrückt, erhalten Sie die aktuellen Mauskoordinaten.

Kapitel: 2.8

MOUSE ON
MOUSE OFF
MOUSE STOP

Syntax: -

Diese Befehle aktivieren oder deaktivieren Event Trapping für die Maus. Ein Mausklick ist ein Ereignis, das durch Event Trapping überprüft werden kann. MOUSE ON ermöglicht Mouse Trapping, MOUSE OFF schaltet es ab, und MOUSE STOP unterdrückt es bis zum nächsten MOUSE ON. Das Unterprogramm, das sich um den Mausklick kümmert, können Sie mit ON MOUSE GOSUB angeben.

Kapitel: 2.8

NAME

Syntax: NAME (Alter Dateiname) AS (Neuer Dateiname)

Mit NAME benennen Sie eine Datei auf Diskette, Festplatte oder RAM-Disk um. Bitte verwenden Sie im neuen Namen keine andere Geräte- oder Schubladenbezeichnung als im alten Namen, sonst erhalten Sie den Error "Rename across disks".

Kapitel: 3.2

NEW

Syntax: -

Mit NEW löschen Sie das Programm, das zur Zeit im Speicher steht. Vorher werden alle Dateien geschlossen und mit TROFF der Trace-Modus abgeschaltet. Wenn Sie das Programm in der aktuellen Form noch nicht abgespeichert haben, weist Sie ein Dialog-Window darauf hin.

Kapitel: 1.9

NEXT

siehe FOR...NEXT

NOT

Syntax: **Wert = NOT Wert**

Die logische Verknüpfung NOT macht aus 0 den Wert -1 und aus -1 den Wert 0. Er kehrt also die Richtigkeit einer Bedingung um:

```
IF NOT (a<1) THEN ...
```

THEN wird ausgeführt, wenn a **nicht** kleiner 1 ist, also größer oder gleich 1. Um die Umkehrung zu erreichen, wird der angegebene Wert nach folgender Formel umgerechnet:

```
NOT x = -(x+1)
```

Für alle anderen 'x'-Werte außer -1 und 0 macht das allerdings nicht viel Sinn.

Kapitel: Zwischenspiel 4

OBJECT.AX und *OBJECT.AY*

Syntax: OBJECT.AX Objektnummer,Beschleunigung
OBJECT.AY Objektnummer,Beschleunigung

Mit diesen beiden Befehlen geben Sie eine Beschleunigung für die Objekt-Bewegung an. Mit OBJECT.AX legen Sie den horizontalen Beschleunigungswert fest. Positive Zahlen beschleunigen das Objekt von links nach rechts, negative Zahlen beschleunigen es von rechts nach links.

OBJECT.AY gibt die vertikale Beschleunigung an. Positive Werte beschleunigen das Objekt von oben nach unten, negative Werte von unten nach oben. Schauen Sie sich zu diesem Thema bitte Bild 4 in Kapitel 1.15 an.

Die Beschleunigung wird in Bildpunkten pro Sekunde angegeben. Die von Ihnen programmierte Bewegung beginnt aber erst durch den OBJECT.START-Befehl.

Kapitel: 1.15

OBJECT.CLIP

Syntax: OBJECT CLIP (x1,y1)-(x2,y2)

Mit OBJECT.CLIP legen Sie ein Rechteck fest. AmigaBASIC wird Objekt-Bewegungen nur innerhalb dieses Rechtecks durchführen. Das heißt: Wenn die Koordinaten des Objekts innerhalb des Clip-Bereich liegen, ist das Objekt sichtbar. Liegen sie außerhalb, ist das Objekt unsichtbar. Als Standardwert nimmt AmigaBASIC die Ausmaße des Ausgabewindows an. Wird das Ausgabewindow verkleinert oder vergrößert, ändert sich der Clip-Bereich aber nicht automatisch mit!

OBJECT.CLOSE

Syntax: OBJECT.CLOSE [Objektnummer] [,Objektnummer, ...]

Solange ein Objekt im Speicher ist, benötigt es Speicherplatz. Wird ein Objekt also nicht mehr gebraucht, sollten Sie mit OBJECT.CLOSE den besetzten Speicherplatz freigeben. Geben Sie keine 'Objektnummer' an, werden alle Objekte, die ins aktuelle Ausgabefenster gehören, gelöscht und ihr Speicherplatz wird freigegeben.

OBJECT.HIT

Syntax: OBJECT.HIT Objektnummer, [Wert1] [,Wert2]

Dieser Befehl legt für jedes Objekt einzeln fest, mit welchen anderen Objekten (oder Bildschirmbegrenzungen) es kollidieren kann. So können Sie Kollisionen zwischen bestimmten Objekten unterdrücken (z.B. einzelne Objekte mit dem Bildschirmrahmen oder Raumschiff und Sterne in einem Spielprogramm). Ist eine bestimmte Kollision möglich, und sie findet statt, ist das ein Ereignis, das durch Collision-Event-Trapping überprüft werden kann. Näheres bei ON COLLISION GOSUB.

Die Standardeinstellung ist, daß alles mit jedem zusammenstoßen kann. Also Objekte untereinander und jedes Objekt mit dem Bildschirmrahmen. Für 'Wert1' und 'Wert2' können Sie jeweils einen 16-Bit-Wert als Dezimalzahl angeben. Ist das niedrigste Bit in 'Wert2' gesetzt (ist 'Wert2' also ungerade), kann das angegebene Objekt mit dem Bildschirmrand zusammenstoßen. Ist dieses Bit nicht gesetzt, kann das Objekt über den Bildschirmrahmen hinausfliegen.

Die restlichen Bits werden folgendermaßen verwendet: Wert1 steht für das Objekt selbst. 'Wert2' steht für andere Objekte, mit denen unser Objekt zusammenstoßen könnte. Tritt eine Kollision ein, verknüpft AmigaBASIC den 'Wert2' des fremden Objekts

mit dem 'Wert1' unseres Objekts durch AND. Kommt dabei 0 heraus, stimmt also kein Bit in den beiden Werten überein, findet keine Kollision statt. Bei jedem anderen Ergebnis kracht's.

Ein Beispiel:

Objekt	Wert1	Wert2
Raumschiff	00000000 00000010	00000000 00000011
Rakete	00000000 00000010	00000000 00000010
Sterne	00000000 00000100	00000000 00000100

Wir haben zum besseren Vergleich die 16-Bit-Zahl binär dargestellt. Im OBJECT.HIT-Befehl müßten Sie den Wert als Dezimalzahl angeben. Der Befehl fürs Raumschiff lautet also z.B.:

```
OBJECT.HIT 1,2,3
```

Nun zu den Bits: Das niedrigste Bit in 'Wert2' des Raumschiffs ist gesetzt. Das Raumschiff kann also nicht außerhalb des Rahmens fliegen. Die Rakete und die Sterne dürfen den Bildschirm in alle vier Richtungen verlassen.

In 'Wert1' des Raumschiffs und 'Wert2' der Rakete ist das zweite Bit gesetzt, die beiden können also kollidieren. Genauer gesagt: Trifft die Rakete auf das Raumschiff, gibt es eine Kollision. Damit es auch kracht, wenn umgekehrt das Raumschiff auf die Rakete fliegt, haben wir auch das zweite Bit in 'Wert2' des Raumschiffs und in 'Wert1' der Rakete gesetzt. Für die Sterne ist nur das dritte Bit gesetzt, sie sind an keiner Kollision beteiligt. Zumindest nicht mit dem Raumschiff oder der Rakete.

OBJECT.ON und OBJECT.OFF

Syntax: OBJECT.ON [Objektnummer] [,Objektnummer, ...]
OBJECT.OFF [Objektnummer] [,Objektnummer, ...]

OBJECT.ON macht die angegebenen Objekte sichtbar, OBJECT.OFF macht sie unsichtbar. Geben Sie gar keine 'Objektnummer' an, bezieht sich der Befehl auf alle Objekte im aktuellen Ausgabewindow. OBJECT.OFF sorgt auch dafür, daß ein Objekt, das sich bewegt, angehalten wird. So gibt es keine Kollisionen mit Geisterfliegern.

Kapitel: 1.15

OBJECT.PLANES

Syntax: OBJECT.PLANES Objektnummer [,Bitebenen] [,Ebenenwert]

Diesen Befehl werden Sie wahrscheinlich recht selten benutzen, da er ziemlich tief im Speicher des Amiga herumbastelt: Der Wert 'Bitebenen' ist ein als Dezimalzahl geschriebener 8-Bit-Wert. Die gesetzten Bits legen fest, in welcher Bitebene das Objekt erscheinen soll. Haben Sie z.B. vier Bitebenen zur Verfügung und wollen das Objekt in der ersten und der dritten Ebene haben (Ebene 0 und Ebene 3), müssen Sie für 'Bitebenen' angeben: $2^0 + 2^2 = 1+4 = 5$.

Der andere Wert, 'Ebenenwert' (ebenfalls ein 8-Bit-Wert), gibt an, was in den anderen Bitebenen dargestellt werden soll. Ist das Bit für die jeweilige Ebene gesetzt, wird der Bob in dieser Ebene mit 1er-Bits aufgefüllt. Ist das Bit nicht gesetzt, erscheinen in dieser Ebene an der Stelle des Bobs 0er-Bits.

OBJECT.PRIORITY

Syntax: **OBJECT.PRIORITY Objektnummer, Prioritätswert**

Mit OBJECT.PRIORITY legen Sie die Priorität einzelner Objekte fest. Das heißt, wir klären die Frage "Welches Objekt wird zuerst dargestellt, wenn mehrere übereinander liegen?" Als 'Prioritätswert' vergeben Sie jedem Objekt eine Zahl zwischen -32768 und +32767. Von zwei Objekten wird das Objekt im Vordergrund abgebildet, das den höheren Prioritätswert hat. Bei gleichem Prioritätswert entscheidet das Los. Oder um genauer zu sein: Es hängt vom Zufall ab, wer zuerst darf.

OBJECT.SHAPE

Syntax: **OBJECT.SHAPE Objektnummer, Definitionsstring**
OBJECT.SHAPE Objektnummer, andere Objektnummer

Dieser Befehl legt die Form, die Farben und die Größe eines Objekts fest. Alle diese Daten stehen in einem Definitionsstring, den Sie aus einer sequentiellen Datei einlesen können:

```
OPEN "(Objektdatei)" FOR INPUT AS 1
  OBJECT.SHAPE Objektnummer, INPUT$(LOF(1),1)
CLOSE 1
```

Die Daten in der sequentiellen Objektdatei können Sie mit dem Programm "ObjectEditor" aus der "BASICDemos"-Schublade der "Extras"-Diskette erzeugen. Näheres dazu im Kapitel 1.11.

Die zweite gezeigte Syntax ermöglicht Ihnen, die Definition eines bereits eingelesenen Objekts auf ein anderes Objekt zu kopieren. Dann haben Sie zwei oder mehrere gleiche Objekte im Speicher.

Kapitel: 1.13

OBJECT.START und *OBJECT.STOP*

Syntax: **OBJECT.START [Objektnummer] [,Objektnummer,...]**
OBJECT.STOP [Objektnummer] [,Objektnummer,...]

OBJECT.START startet die festgelegte Bewegung eines Objekts, OBJECT.STOP hält sie an. Sie können auch mehrere Objekte gleichzeitig in Bewegung setzen oder anhalten. MIT OBJECT.VX und .VY oder OBJECT.AX und .AY legen Sie eine Geschwindigkeit und/oder eine Beschleunigung fest. Mit OBJECT.X und OBJECT.Y geben Sie die Ausgangsposition an. Durch OBJECT.START beginnt die Bewegung. Sie läuft von nun an im Hintergrund ab, Ihr Programm kann sich um andere Dinge kümmern. Bei einer Kollision führt AmigaBASIC für alle beteiligten Objekte automatisch OBJECT.STOP aus.

Kapitel: 1.15

OBJECT.VX und *OBJECT.VY*

Syntax: **OBJECT.VX Objektnummer, Geschwindigkeitswert**
OBJECT.VY Objektnummer, Geschwindigkeitswert

Diese beiden Befehle sind für die Festlegung einer Geschwindigkeit zuständig. Mit OBJECT.VX geben Sie den horizontalen Geschwindigkeitswert an, mit OBJECT.VY den vertikalen Anteil.

Positive Werte erzeugen eine Bewegung von rechts nach links bzw. von oben nach unten. Für Bewegungen von links nach rechts bzw. unten nach oben verwenden Sie einfach negative Geschwindigkeitswerte. Schauen Sie sich zu diesem Thema bitte Bild 4 in Kapitel 1.15 an. Die Geschwindigkeit geben Sie in Pixels pro Sekunde an. Die definierte Bewegung beginnt erst durch den OBJECT.START-Befehl.

Kapitel: 1.15

OBJECT.VX und *OBJECT.VY*

Syntax: Wert = OBJECT.VX (Objektnummer)
Wert = OBJECT.VY (Objektnummer)

Diese Funktion teilt Ihnen die momentane Geschwindigkeit des angegebenen Objekts mit. So können Sie diese Information für Berechnungen verwenden, z.B. um die Geschwindigkeit von Zeit zu Zeit zu ändern.

Kapitel: 1.15

OBJECT.X und *OBJECT.Y*

Syntax: OBJECT.X Objektnummer, x-Koordinate
OBJECT.X Objektnummer, y-Koordinate

Mit OBJECT.X und OBJECT.Y legen Sie die Position eines Objekts oder den Ausgangspunkt seiner Bewegung fest. Die x- und die y-Koordinate geben die Lage der linken oberen Ecke des Objekts an. Die Werte, die das Objekt auf dem Bildschirm sichtbar machen, hängen von der Auflösungsstufe des aktuellen Windows ab.

Objekte dürfen sich aber auch außerhalb des sichtbaren Bereichs befinden, die Koordinaten müssen zwischen -32768 und +32767 liegen. Bedenken Sie, daß Objekte erst nach OBJECT.ON sichtbar sind.

Kapitel: 1.15

OBJECT.X und *OBJECT.Y*

Syntax: **Wert = OBJECT.X (Objektnummer)**
Wert = OBJECT.Y (Objektnummer)

Diese Version von OBJECT.X und OBJECT.Y arbeitet als BASIC-Funktion: Sie können den gegenwärtigen Aufenthaltsort des gewünschten Objekts abfragen. Anhand dieser Information kann Ihr Programm entscheiden, was mit dem Objekt geschehen soll, ohne sich ständig darum kümmern zu müssen. Das Ergebnis gibt die Bildschirmkoordinaten der linken oberen Ecke des Objekts an.

Kapitel: 1.15

OCT$

Syntax: **String=OCT$(Wert)**

Diese Funktion liefert die oktale Schreibweise einer Zahl zwischen -32768 und 65535. Wenn Sie sich nicht mehr daran erinnern, was eine oktale Zahl sein soll, lesen Sie bitte Zwischenspiel 4.

OCT$(60037) ergibt 165205.

ON BREAK GOSUB

Syntax: **ON BREAK GOSUB Label**

Mit diesem Befehl teilen Sie AmigaBASIC das Unterprogramm mit, das aufgerufen wird, wenn ein Programmabbruch auftritt und Event Trapping für dieses Ereignis aktiv ist. Sie können ein Label oder eine Zeilennummer angeben. Geben Sie jedoch als Label die Zahl 0 an, wird damit das Abbruch-Event-Trapping abgeschaltet. Zum Aktivieren müssen Sie außerdem den Befehl BREAK ON geben.

Im Unterprogramm sollten Sie den Anwender darauf hinweisen, daß das Programm nicht abgebrochen werden kann, und ihm gegebenenfalls auch mitteilen, wie er das Programm trotzdem beenden kann. (Schön wäre doch z.B. ein Window "Wollen Sie das Programm wirklich beenden?" mit einem Ja- und einem Nein-Feld.)

Während das Unterprogramm ausgeführt wird, blockiert AmigaBASIC das Break Trapping, damit nicht ein zweites Ereignis die Bearbeitung des ersten unterbricht. Das Unterprogramm muß mit RETURN beendet werden.

ON COLLISION GOSUB

Syntax: ON COLLISION GOSUB Label

Mit diesem Befehl steuern Sie Event Trapping für die Kollisionsüberwachung. Sie legen fest, welches Unterprogramm bei einer Kollision aufgerufen werden soll. Geben Sie einfach das Label (die Sprungmarke) der zuständigen Routine an.

Wenn Sie anstelle eines Labels (oder einer Zeilennummer) den Wert 0 angeben, wird Event Trapping für Kollisionen abgeschaltet. Selbst dann, wenn Sie '0' als Label oder Zeilennummer verwendet haben. Um Collision-Event-Trapping zu aktivieren, brauchen Sie außerdem noch den Befehl COLLISION ON.

ON ERROR GOSUB

Syntax: ON ERROR GOSUB Label

Damit erfährt AmigaBASIC, welches Unterprogramm aufgerufen werden soll, falls ein Fehler auftritt und das Event Trapping für Errors aktiv ist. Sie können ein Label oder eine Zeilennummer angeben. Geben Sie jedoch als Label die Zahl 0 an, wird damit Event Trapping für Errors abgeschaltet.

Im Unterprogramm können Sie aus der Variablen ERR die Fehlernummer und aus ERL die letzte Zeilennummer vor der Fehlerzeile erfahren. Mit dem ERROR-Befehl ist es sogar möglich, im Programm eigene Fehlernummern zu benutzen. Tritt im Unterprogramm ein weiterer Fehler auf, bricht AmigaBASIC das Programm ab. Um ins Hauptprogramm zurückzuspringen, müssen Sie am Ende des Unterprogramms den Befehl RESUME verwenden. Weiteres lesen Sie bitte dort.

ON...GOSUB

Syntax: **ON x GOSUB Label [,Label] [,Label]**

Mit diesem Befehl können Sie abhängig vom Wert von 'x' verschiedene Unterprogramme aufrufen lassen. Ist 'x'=1, springt das Programm zum ersten Label, ist 'x'=2, zum zweiten usw. Wenn 'x' einen Nachkommateil hat, wird der Wert auf eine ganze Zahl gerundet. Ist 'x' größer als die Anzahl der angegebenen Labels, macht das Programm in der Zeile hinter ON...GOSUB weiter. Negative 'x' erzeugen einen 'Illegal function call'-Error.

Kapitel: 2.9

ON...GOTO

Syntax: **ON x GOTO Label [,Label] [,Label]**

Bitte lesen Sie die Erklärungen zu ON...GOSUB. ON...GOTO funktioniert genauso, nur wird hier kein Unterprogramm aufgerufen, sondern zum angegebenen Programmteil verzweigt.

Kapitel: 2.9

ON MENU GOSUB

Syntax: ON MENU GOSUB Label

Mit diesem Befehl teilen Sie AmigaBASIC das Unterprogramm mit, das aufgerufen wird, wenn der Anwender einen Punkt aus einem Pulldown-Menü auswählt und Event Trapping für dieses Ereignis aktiv ist. Sie können ein Label oder eine Zeilennummer angeben. Geben Sie jedoch als Label die Zahl 0 an, wird damit Event Trapping für Pulldown-Menüs abgeschaltet. Zum Aktivieren müssen Sie außerdem den Befehl MENU ON geben. Im Unterprogramm können Sie die Menü-Auswahl mit der Funktion MENU(x) abfragen. Während das Unterprogramm ausgeführt wird, blockiert AmigaBASIC das Menu Trapping, damit nicht ein zweites Ereignis die Bearbeitung des ersten unterbricht. Das Unterprogramm muß mit RETURN beendet werden.

Kapitel: 2.8

ON MOUSE GOSUB

Syntax: ON MOUSE GOSUB Label

Mit diesem Befehl teilen Sie AmigaBASIC das Unterprogramm mit, das aufgerufen wird, wenn der Anwender mit der linken Maustaste klickt und Event Trapping für dieses Ereignis aktiv ist. Sie können ein Label oder eine Zeilennummer angeben. Geben Sie jedoch als Label die Zahl 0 an, wird damit Event Trapping für die Maus abgeschaltet. Zum Aktivieren müssen Sie außerdem den Befehl MOUSE ON geben. Im Unterprogamm können Sie Informationen über die Maus mit der Funktion MOUSE(x) abfragen. Während das Unterprogramm ausgeführt wird, blockiert AmigaBASIC das Mouse Trapping, damit nicht ein zweites Ereignis die Bearbeitung des ersten unterbricht.

Das Unterprogramm muß mit RETURN beendet werden.

Kapitel: 2.8

ON TIMER GOSUB

Syntax: ON TIMER(x) GOSUB Label

Mit diesem Befehl teilen Sie AmigaBASIC das Unterprogramm mit, das aufgerufen wird, wenn seit dem letzten Timer-Event 'x' Sekunden vergangen sind und Event Trapping für dieses Ereignis aktiv ist. Diese Version von Event Trapping ermöglicht den Aufruf eines Unterprogramms in regelmäßigen Zeitabständen. 'x' muß zwischen 1 (1 Sekunde) und 86400 (60*60*24 Sekunden = 24 Stunden) liegen. Sie können ein Label oder eine Zeilennummer angeben. Geben Sie jedoch als Label die Zahl 0 an, wird damit das Timer-Event-Trapping abgeschaltet. Zum Aktivieren müssen Sie außerdem den Befehl TIMER ON geben.

Während das Unterprogramm ausgeführt wird, blockiert AmigaBASIC das Timer Trapping, damit nicht ein zweites Ereignis die Bearbeitung des ersten unterbricht. Das Unterprogramm muß mit RETURN beendet werden.

OPEN

Syntax: OPEN Dateiname [FOR Modus] AS [#] Dateinummer [LEN=Länge]
OPEN KurzModus, [#] Dateinummer,Dateiname [,Länge]

Der OPEN-Befehl öffnet eine Datei. Sie können sich eine der beiden Syntax-Versionen aussuchen, sie bewirken beide dasselbe. In der ersten Syntax-Version geben Sie zuerst den Dateinamen an, dann den Modus: Sie können eine Datei FOR INPUT (zum Lesen), FOR OUTPUT (zum Schreiben) oder FOR APPEND (zum Anhängen von Daten) öffnen. Alle diese Modi arbeiten mit einer sequentiellen Datei. Wenn Sie FOR und den Modus bei der ersten Syntax-Variante weglassen, öffnet AmigaBASIC eine relative Datei.

Am Ende der OPEN-Zeile können Sie optional die Puffergröße in Bytes angeben. Bei relativen Dateien ist das sehr wichtig: Die Zahl, die Sie hier angeben müssen, ist die Summe der Feldlängen aus dem FIELD-Befehl. Die zweite Syntax sieht etwas

anders aus, funktioniert aber genauso: 'KurzModus' ist ein Buchstabe, der den Dateimodus angibt: I für sequentiellen Input, O für sequentiellen Output, A für sequentiellen Append-Modus und R für eine relative Datei. Die anderen Angaben entsprechen genau der ersten Syntax-Version.

Sie können sequentielle Dateien zu allen Ein- und Ausgabegeräten öffnen (SCRN:, KYBD:, PRT:, PAR:, SER:) und von dort Daten lesen bzw. dorthin Daten schicken. Eine sequentielle Datei darf nur entweder zum Lesen oder zum Schreiben geöffnet werden, beides gleichzeitig geht nicht.

Kapitel: 3.3

OPEN "COM1:"

Syntax: **OPEN "COM1: [Baud] [,Parität] [,Wortlänge] [,Stopbits]" [FOR Modus] AS [#] [Dateinummer]**

Mit diesem Befehl, einer Variante des OPEN-Befehls, öffnen Sie eine Datei zur seriellen Schnittstelle. Dort können Sie Geräte mit einem sogenannten RS232-Interface betreiben, einer Schnittstelle zur seriellen Datenübertragung. Wenn Sie kein solches Interface benutzen, werden Ihnen die folgenden Informationen nicht viel sagen und auch nicht viel helfen. Sie brauchen diesen Befehl dann nicht kennenzulernen. Der Wert 'Baud' gibt die Übertragungsgeschwindigkeit an. 1 Baud ist 1 Bit/Sekunde. Mögliche Werte sind 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600 und 19200. Der voreingestellte Baud-Wert ist 9600. Welchen Wert Sie verwenden müssen, hängt von dem Gerät ab, mit dem Sie Daten austauschen wollen.

'Parität' legt fest, nach welchem Modus die Daten beim Senden und Empfangen überprüft werden. Auch dieser Wert muß bei Sender und Empfänger einheitlich sein. Sie können einen von drei Buchstaben angeben: E (engl.: even) bedeutet "gerade Bit-

Parität". O (engl.: odd) bedeutet "ungerade Bit-Parität". N (engl.: none) bedeutet "keine Paritätsprüfung". Die Grundeinstellung ist E.

'Wortlänge' gibt an, wieviel Bits pro Byte Informationen enthalten. Möglich sind: 5,6,7 und 8, der Standardwert ist 7.

'Stopbits' legt die Anzahl an Stopbits fest. Auch dieser Wert muß bei serieller Übertragung geregelt werden. Zur Auswahl stehen 1 oder 2. Bei 110 Baud sind 2 Stopbits voreingestellt, bei allen anderen Baud-Raten 1 Stopbit. Sie können diesen Wert natürlich nach Belieben ändern.

Als Modus können Sie FOR INPUT oder FOR OUTPUT wählen. Sie können den Modus auch weglassen, dann wird ein Modus benutzt, in dem Sie gleichzeitig senden und empfangen können.

OPTION BASE

Syntax: OPTION BASE x

Beim Dimensionieren von Datenfeldern hat jedes Feld als niedrigstes Element das Feldelement Nummer 0 (Hallo(0), Farben(0,0)). Wenn Sie wollen, daß alle Felder mit dem Element 1 beginnen (Hallo(1), Farben(1,1)), benutzen Sie den Befehl OPTION BASE 1. Diese Befehl muß allerdings vor allen DIMs und vor der Benutzung von Feldelementen gegeben werden. Sonst erhalten Sie einen "Duplicate definition"-Error.

OR

Syntax: **Wert = Wert1 OR Wert2**

Die logische Verknüpfung OR verknüpft die einzelnen Bits von 'Wert1' und 'Wert2' folgendermaßen:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

Kapitel: Zwischenspiel 4

PAINT

Syntax: **PAINT [STEP] (x,y) [,Ausmalfarbe] [,Begrenzungsfarbe]**

Dieser Befehl malt ein geschlossene Fläche in der angegebenen 'Ausmalfarbe' aus. Begrenzungen erkennt er nur, wenn sie in der 'Begrenzungsfarbe' gemalt sind. Haben Sie keine 'Begrenzungsfarbe' angegeben, gilt die 'Ausmalfarbe' auch gleichzeitig als 'Begrenzungsfarbe'. Fehlen beide Werte, arbeitet AmigaBASIC mit der aktuellen Zeichenfarbe. Ein Window, in dem PAINT benutzt werden soll, muß einen Windowtyp zwischen 16 und 31 haben (d.h.: Der Windowinhalt wird im Speicher gepuffert).

Kapitel: 2.6

PALETTE

Syntax: **PALETTE Farbnummer, Rot-Anteil, Grün-Anteil, Blau-Anteil**

Mit PALETTE können Sie eigene Farben erzeugen. Die Anzahl der möglichen Farben richtet sich nach der Anzahl der Bitebenen des Screens, auf dem sich das Ausgabewindow befindet. Jede verwendete Farbe hat eine Farbnummer. Dieser Nummer

(im Höchstfall zwischen 0 und 31) wird dann eine bestimmte Farbe zugeordnet.

Für den 'Rot-Anteil', den 'Grün-Anteil' und den 'Blau-Anteil' verwenden Sie eine Zahl zwischen 0 (entspricht 0%, kein Anteil) und 1 (entspricht 100%, voller Anteil). Die neue Farbeinstellung ersetzt die alte Einstellung. Lediglich die vier Workbenchfarben (also die Farben 0 bis 3, in denen auch die Workbench arbeitet, und die mit Preferences geändert werden können) werden von Amiga beim Starten und Beenden eines Programms zurückgesetzt.

Kapitel: 1.16

PATTERN

Syntax: **Pattern [16-Bit-Wert für Linien] [,Feld für Flächen]**

Mit diesem Befehl legen Sie die Füllmuster für Linien oder Flächen fest. Für Linien können Sie einen 16-Bit-Wert angeben. Damit definieren Sie ein 16 Pixels breites Muster. Jedes gesetzte Bit in der Zahl entspricht einem Punkt auf dem Bildschirm. Den 16-Bit-Wert können Sie als Dezimalzahl (z.B. 65535), Hexadezimalzahl (&HFFFF) oder Oktalzahl (&O177777) angeben.

Für Flächen geben Sie ein Datenfeld mit Integerzahlen an. Die Zahlen in diesem Feld definieren ein Muster, das für sämtliche Füll- und Ausmal-Aktionen (PAINT, AREAFILL oder LINE...,BF) verwendet wird. Auch hier entsprechen die gesetzten Bits sichtbaren Punkten auf dem Bildschirm. Die Anzahl der Feldelemente im Integerfeld muß eine ganze Zweierpotenz sein. (Also 1, 2, 4, 8, 16 oder 32 Elemente usw.)

Kapitel: 2.10

PEEK

Syntax: `Wert=PEEK(Adresse)`

PEEK liefert den Inhalt der Speicherzelle an der angegebenen Adresse. Das ist eine Zahl zwischen 0 und 255, ein Byte. Die höchste erlaubte Adresse im Amiga ist 16777215. Viele der möglichen Adressen sind aber gar nicht mit einer Speicherzelle verbunden, solange Sie den Speicher nicht voll auf 8 Megabyte RAM ausgebaut haben.

Kapitel: 4.3

PEEKL

Syntax: `Wert=PEEKL(Adresse)`

PEEKL liefert den Inhalt von vier aufeinanderfolgenden Speicherzellen ab einer geraden Adresse. Das Ergebnis ist eine Zahl zwischen -2147483648 und 2147483647, eine 4-Byte-Zahl. Über die Adresse erfahren Sie mehr bei PEEK.

Kapitel: 4.3

PEEKW

Syntax: `Wert=PEEKW(Adresse)`

PEEKL liefert den Inhalt von zwei aufeinanderfolgenden Speicherzellen ab einer geraden Adresse. Das Ergebnis ist eine Zahl zwischen -32768 und 32767, eine 2-Byte-Zahl. Über die Adresse erfahren Sie mehr bei PEEK.

Kapitel: 4.3

POINT

Syntax: **Wert=POINT(x,y)**

Diese BASIC-Funktion liefert die Farbnummer der Farbe eines angegebenen Punkts im aktuellen Ausgabewindow. Liegt dieser Punkt außerhalb des Windows, erhalten Sie das Ergebnis -1.

POKE

Syntax: **POKE Adresse,Wert**

POKE schreibt einen 1-Byte-Wert in die Speicherzelle mit der angegebenen Adresse. Über die Adresse erfahren Sie mehr bei PEEK.

Beim Schreiben von Werten in den Speicher sollten Sie vorsichtig sein. Ohne genaue Planung und detaillierte Kenntnisse über das Amiga-Betriebssystem richten Sie sicher mehr Schaden als Nutzen an.

Kapitel: 4.3

POKEL

Syntax: **POKEL Adresse,Wert**

POKE schreibt einen 4-Byte-Wert in vier aufeinanderfolgende Speicherzellen ab einer geraden Adresse. Die möglichen Adressen finden Sie bei PEEK. Weiteres bei PEEKL und POKE.

Kapitel: 4.3

POKEW

Syntax: **POKEW Adresse,Wert**

POKE schreibt einen 2-Byte-Wert in zwei aufeinanderfolgende Speicherzellen ab einer geraden Adresse. Die möglichen Adressen finden Sie bei PEEK. Weiteres bei PEEKW und POKE.

Kapitel: 4.3

POS

Syntax: **Wert=POS(x)**

Für 'x' dürfen Sie jeden beliebigen Wert angeben. Es dient bloß als Schein-Argument. (Das hat nichts mit politischen Diskussionen zu tun, sondern damit, daß viele BASIC-Funktionen ein Argument, also einen Wert in Klammern, haben müssen.) POS(x) liefert die Spalte, in sich der Cursor zur Zeit befindet. Mit "Cursor" meinen wir den Ausgabecursor, also die Bildschirmposition, an der das nächste Zeichen erscheint. Wenn Sie die aktuelle Cursorposition benötigen, geht das so:

```
Zeile=CSRLIN : Spalte=POS(0)
```

PRESET und PSET

Syntax: **PRESET [STEP] (x,y) [Farbe]**
PSET [STEP] (x,y) [,Farbe]

Diese beiden Befehle zeichnen einen Punkt ins aktuelle Ausgabewindow. 'x' und 'y' geben die Koordinaten des Punkts an. Mit STEP können Sie die Koordinaten als Abstand zum letzten Punkt angeben. 'Farbe' ermöglicht die Angabe einer Farb-

nummer, um die Zeichenfarbe festzulegen. Zwischen PRESET und PSET gibt es nur einen Unterschied: Wenn Sie keine Zeichenfarbe angeben, zeichnet PRESET in der aktuellen Hintergrundfarbe, wogegen PSET die aktuelle Zeichenfarbe verwendet.

Kapitel: 2.5

PRINT

Syntax: **PRINT [Variable oder Wert] [Trennzeichen] [Variable oder Wert]...**

Der PRINT-Befehl kann durch ? abgekürzt werden. Er dient zur Ausgabe von Variablen oder Werten (auch Stringvariablen oder Texten) auf dem Bildschirm. Texte müssen in Anführungszeichen angegeben werden. Geben Sie überhaupt keine Variablen oder Werte an, wird eine Leerzeile gedruckt. Als Trennzeichen kommen verschiedene Zeichen in Frage: Ein Strichpunkt (;) bewirkt, daß die einzelnen Angaben direkt hintereinander gedruckt werden. Ein Komma (,) bewirkt einen Sprung zur nächsten Tabulatorposition (alle 15 Spalten, die Tabulatoreinstellung kann mit WIDTH verändert werden). Um Variablennamen voneinander zu trennen, dürfen Sie auch das Leerzeichen und die Sonderzeichen # & und % verwenden. Sie wirken in diesem Fall wie der Stichpunkt. Steht ein Stichpunkt am Ende, beginnt die nächste Druckausgabe unmittelbar hinter der letzten. Anderenfalls wird ein Zeilenvorschub ausgeführt.

Kapitel: 1.3

PRINT#
PRINT# ,USING

Syntax: **PRINT # Dateinummer [USING (FormatString);] [[Variable oder Wert] [Trennzeichen] [Variable oder Wert ...]**

Diese beiden Befehle funktionieren genauso wie PRINT und PRINT USING, aber sie schreiben ihre Ausgaben in eine Datei. Näheres bei PRINT, PRINT USING und INPUT#.

Kapitel: 3.3

PRINT USING

Syntax: **PRINT USING (FormatString); (Variable oder Wert) [;]**

PRINT USING dient zur formatierten Ausgabe von Werten. So können Sie zum Beispiel verschiedene Zahlen in Tabellen drukken, wobei das Komma immer auf derselben Position steht. Das ist natürlich auch für Stringvariablen möglich. In einem Formatstring geben Sie Ihre Formatierungswünsche an. Schauen wir mal, was da alles möglich ist:

Ein Doppelkreuz # (das amerikanische Zeichen für "Nummer") steht für eine Ziffer. Ein Punkt steht für den amerikanischen Dezimalpunkt. In Deutschland stünde hier das Komma. Der Nachkommateil, der nicht gedruckt wird, wird gerundet.

```
print using "###.##";32.4        ergibt    32.40
print using "###.##";17                    17.00
print using "###.##";324.124               324.12
print using "###.##";128.489               128.49
print using "###.##";129.9984              130.00
```

Hat die zu druckende Zahl mehr Stellen vor dem Komma, als #-Positionen angegeben wurden, erscheint vor der Zahl ein %-Zeichen zur Kennzeichnung.

```
print using "####.###";43259.3253          %43259.325
```

Ein + am Anfang des Formatstrings bewirkt die Angabe des Vorzeichens vor dem Wert. Das + kann auch am Ende stehen, dann erscheint das Vorzeichen hinter dem Wert. Am Ende des Formatstrings ist auch ein - erlaubt. Dadurch wird dann hinter negativen Werten ein Minuszeichen gedruckt.

```
print using "+###.##";324.234              +324.23
print using "+###.##";-518.284             -518.28
print using "###.##+";-518.294             518.28-
print using "###.##-";324.234              324.23
print using "###.##-";-518.294             518.27-
```

Wenn Sie zwei Sterne (**) anstelle der ersten beiden Nummernzeichen (#) angeben, werden führende Leerstellen mit Sternen aufgefüllt. Die beiden * zählen als Ziffernpositionen mit.

```
print using "**##.##";42.2                 **42.20
print using "**#####.##";43.1              *****43.10
print using "**###.##";-523.456            *-523.46
```

Zwei Dollarzeichen ($$) bewirken, daß vor der Zahl ein $ gedruckt wird. Das erste Dollarzeichen steht für das später gedruckte Zeichen, das zweite steht für eine Ziffernposition. Wollen Sie mit + das Vorzeichen in jedem Fall drucken, muß es hinter der Zahl stehen.

```
print using "$$###.##";43.54               $43.54
print using "$$###.##";-43.54              -$43.54
print using "$$###.##";-53245.345          %-$53245.35
print using "$$###.##+";45.3               $45.30+
print using "$$+##.##";45.3                %$45+
```

Das letzte Beispiel zeigt also einen Fehler: + darf nicht in Verbindung mit $$ angegeben werden. Wie Sie sehen, kommt PRINT USING dann nämlich durcheinander und schneidet kurzerhand den Nachkommateil ab.

Sie können aber den Effekt von ** und $$ kombinieren. Dazu verwenden Sie **$. Diese Angabe zählt als drei Zeichenpositionen. Eine für das Dollarzeichen und zwei für Ziffern bzw. Sterne. Das sieht dann so aus:

```
print using "**$##.##";45.3          **$45.30
print using "**$##.##";345.3         *$345.30
```

Gerade bei Sternchen und Dollars wird natürlich klar, daß PRINT USING besonders für amerikanische Verhältnisse und amerikanische Schreibweisen geeignet ist. So auch der nächste Fall: Die Benutzung von Kommas und Punkten bei Zahlen ist in Amerika genau andersrum als in Deutschland. Wenn wir große Zahlen untergliedern wollen, trennen wir Dreiergruppen mit einem Punkt. In Amerika nimmt man dazu das Komma:

Deutsch: 3.444.233,50 Amerikanisch: 3,444,233.50

Die amerikanische Version wird auch von PRINT USING unterstützt: Ein Komma links vom Dezimalpunkt erreicht diese Einteilung:

```
print using "#########,.##",3444233.50       3,444,233.50
```

Die erzeugten Kommas zählen als Zeichenposition.

Erinnern Sie sich noch an unser Zwischenspiel 7? Besonders an die Exponentialzahlen? Wenn Sie wollen (aber wirklich nur dann), können Sie die auch mit PRINT USING erzeugen. Hängen Sie einfach hinten am Formatstring vier Potenzierungssymbole an: ^^^^

```
print using "##.##^^^^";34.53        3.45E+01
print using "##.##^^^^";.009         9.00E-03
```

Jetzt haben wir's bald geschafft! Als nächstes gibt es noch das Unterstreichungszeichen (_). Es zeigt an, daß das darauffolgende Zeichen nicht als Formatierungssymbol gedacht ist, sondern selbst gedruckt werden soll.

```
print using "_####.##";34.95                # 34.95
```

Sie dürfen auch Text in den Formatierungsstring aufnehmen. Der Text darf dabei beliebig lang sein. Aber weil wir gerade dabei sind: Die Anzahl der definierten Ziffernpositionen im String (also der Stellen, die später von der Zahl benötigt werden) ist auf 24 begrenzt. Wenn Sie also z.B. mehr als 24 #-Zeichen verwenden, gibt's einen "Illegal function call"-Error.

```
print using "Hier die Zahl: ##.##",3.3      Hier die Zahl:  3.30
```

Wenn in Ihrem Text Formatierungszeichen vorkommen, die PRINT USING falsch versteht, schreiben Sie einfach ein _ davor.

Jetzt sind wir schon beim nächsten Thema: Nicht nur Zahlen, auch Strings können formatiert werden. Ein Ausrufezeichen (!) bewirkt, daß nur das erste Zeichen eines Strings gedruckt wird:

```
print using "!";"Amiga"                     A
```

Wenn Sie einen String auf mehr als Zeichen begrenzen wollen, geben Sie die Anzahl als Leerstellen zwischen zwei \-Symbolen (auf der Amiga-Tastatur links neben <BACKSPACE>) ein. Die Anzahl der Leerstellen plus 2 ergibt die Länge des Strings:

```
print using "\  \";"Hallo"                  Hall
```

Zwischen den beiden \ stehen im Beispiel zwei Leerstellen. Ist der String länger, wird er abgeschnitten. Ist er kürzer als die angegebene Stellenzahl, werden Leerstellen angehängt.

Zu guter Letzt schließlich gibt es noch &: Damit werden Strings beliebiger Länge ausdruckt.

```
print using "\\";"Amiga"                    Am
print using "&";"Amiga"                     Amiga
```

Natürlich können Sie das letzte Ergebnis auch ganz ohne den PRINT USING-Befehl erreichen.

Damit hätten wir auch PRINT USING geschafft. Er ist in seiner Vielfalt zugegebenermaßen nicht ganz einfach. Aber wenn Sie die wichtigsten Möglichkeiten kennen, geht die übersichtliche Gestaltung von Listen und Tabellen viel besser als mit LOCATE und den Stringfunktionen. Also experimentieren Sie ruhig noch ein bißchen mit PRINT USING, es lohnt sich.

PSET

siehe PRESET

PTAB

Syntax: PRINT PTAB(x)

Diese Funktion arbeitet genauso wie TAB, nur geben Sie diesmal die Abstände nicht zeichenweise, sondern pixelweise an. So können Sie Text innerhalb der aktuellen Zeile sehr genau positionieren.

Beispiel:

```
FOR x=0 TO 100
  PRINT PTAB(x);"Hallo!"
NEXT x
```

PUT (Grafik)

Syntax: PUT [STEP] (x,y), Feldname [(Feldposition,...)] [,Modus]

Diesen Befehl benötigen Sie, wenn Sie einen Grafikausschnitt, den Sie mit GET in ein Datenfeld eingelesen haben, zurück auf den Bildschirm bringen wollen. Die verschiedenen Parameter

entsprechen dem GET-Befehl (Grafik), bitte vergleichen Sie dort.

Nur der 'Modus' ist hier neu. Sie können verschiedene Arten angeben, wie das Objekt auf dem Bildschirm erscheinen soll: PSET: Die Daten werden genauso auf den Bildschirm gebracht, wie sie ins Feld eingelesen wurden. Die Grafik an der Ausgabestelle wird überschrieben.

PRESET: Das Bild wird invertiert. AND: Nur die Bildpunkte, die nach einer AND-Verknüpfung zwischen dem Bildausschnitt und dem Hintergrund übrigbleiben, werden dargestellt. OR: Feldinhalt und Hintergrund werden durch OR verknüpft; der Ausschnitt wird vollständig in den Hintergrund kopiert.

XOR: Diese Verknüpfung ist die Standardeinstellung des PUT-Befehls. Sie bewirkt, daß das Bild in den Bildschirminhalt übertragen wird. Dort, wo vorher schon etwas war, wird das Bild invertiert.

Kapitel: 4.1

PUT (Daten)

Syntax: PUT [#] Dateinummer [,Satznummer]

Der PUT-Befehl ist bei relativen Dateien dafür zuständig, die Daten aus dem Datensatzpuffer auf Diskette zu schreiben. Erst dann stehen die Daten wirklich in der Datei. Sie können die Satznummer angeben oder auch nicht. Wenn Sie sie nicht angeben, schreibt AmigaBASIC die Daten in den Datensatz, der eine Nummer höher ist als der aktuelle Satz. Wenn Sie die Satznummer angeben, muß sie zwischen 0 und 16777215 liegen. Weiteres dazu beim GET-Befehl (Daten-Version!).

Kapitel: 5.1

RANDOMIZE

Syntax: RANDOMIZE [x]
RANDOMIZE TIMER

Wenn Sie Zufallszahlen (RND) verwenden, haben sie bei jedem Programmdurchlauf denselben Wert. Das ist zwar manchmal ganz praktisch (weil sich "zufällige" Programmsituationen nachvollziehen lassen), aber nicht gerade zufällig. Kein Wunder also, wenn Sie in Ihrem Lottoprogramm immer gewinnen. Abhilfe schafft der RANDOMIZE-Befehl. Mit ihm erreichen Sie, daß verschiedene Zufallszahlen erzeugt werden. Wenn Sie hinter RANDOMIZE eine Zahl zwischen -32768 und 32767 angeben, wird diese als Ausgangswert für die Zufallszahlenberechnung verwendet. Gleiche Zahlen erzeugen auch gleiche Zufallszahlen-Reihen. Geben Sie keinen Wert an, fragt AmigaBASIC mit der Meldung

```
Random Number Seed (-32768 to 32767)?
```

nach einem Ausgangswert. Wenn Sie das nicht wollen, verwenden Sie einfach RANDOMIZE TIMER. AmigaBASIC errechnet dann aus der aktuellen Zeit eine Zufallszahlenreihe. So entstehen wirklich zufällige und bei jedem Programmlauf andere Zahlen.

READ

Syntax: READ Variablenname [,Variablenname,...]

Dieser Befehl liest einen Wert aus einer DATA-Zeile und übergibt diesen Wert der angegebenen Variablen. Die Variable muß vom Variablentyp zum gelesenen Wert passen (Strings zu Strings und Zahlen zu Zahlen). Jeder READ-Befehl erhöht einen internen DATA-Zeiger. Sind alle DATAs gelesen, erhalten Sie die Fehlermeldung 'Out of DATA'.

Kapitel: 2.9

REM

Syntax: **REM [Text]**

Mit REM können Sie Kommentare und Bemerkungen an eine BASIC-Zeile anhängen. Oder dafür auch eine ganze Zeile verwenden. REM-Befehle müssen als letzter Befehl in einer Zeile stehen. Alles, was dahinter kommt, wird von AmigaBASIC überlesen. Sie können anstelle von REM auch ein Hochkomma (') verwenden.

Kapitel: 4.6

RESTORE

Syntax: **RESTORE [Label]**

Wie Sie bei READ und DATA lesen können, erhöht READ einen Zeiger, der auf das nächste zu lesende DATA-Element zeigt. Mit RESTORE setzen Sie diesen Zeiger auf eine angegebene Zeilennummer oder ein Label zurück (oder schieben ihn auch im Programm nach vorne). Geben Sie hinter RESTORE kein Label an, wird der DATA-Zeiger auf den Programmanfang gesetzt.

Kapitel: 4.8

RESUME

Syntax: **RESUME [0]**
RESUME NEXT
RESUME Label

Mit RESUME beenden Sie das Unterprogramm, das sich bei Error Trapping (siehe ON ERROR GOSUB) um die Fehlerbehandlung kümmert. Die erste Syntax-Variante (RESUME oder RESUME 0) führt das Hauptprogramm mit dem Befehl fort, der den Fehler verursacht hat. Die zweite Variante macht mit dem

nächsten Befehl nach dem Fehler-Erzeuger weiter. So können fehlerhafte Befehle einfach ignoriert werden. Die dritte Version erlaubt wie bei RETURN den Rücksprung zu einem beliebigen Label.

RETURN

siehe GOSUB...RETURN

RIGHT$

Syntax: **Teilstring=RIGHT$(String,Länge)**

Diese Stringfunktion liefert einen Teilstring in der angegebenen Länge, der aus dem angegebenen String von der rechten Seite entnommen wird.

RIGHT$("Amiga ist toll!",5) ergibt "toll!"

Kapitel: 1.16

RND

Syntax: **Wert=RND [(x)]**

RND liefert eine Zufallszahl zwischen 0 und 1.

Wenn Sie einen Wert in Klammern angeben, können Sie damit die Erzeugung der Zufallszahlen steuern: 1 und alle positiven Werte erzeugen Zufallszahlen aus einer festen Folge. Bei jedem Programmdurchlauf werden dieselben Zahlen verwendet. Wenn Sie 'x' weglassen, ist ebenfalls dieser Modus aktiv.

0 bringt immer die letzte Zufallszahl. Wenn Sie nur mit RND(0) arbeiten, erhalten Sie ständig dieselbe Zahl. -1 und alle negativen Werte liefern einen neuen Anfangswert für die Zufallszah-

lenerzeugung. Gleiche negative Zahlen ergeben auch die gleiche Zufallszahlenfolge. Lesen Sie deshalb die Erklärungen zum RANDOMIZE-Befehl.

Oft brauchen Sie Zufallszahlen in einem bestimmten Bereich. Hier die Formel dazu:

Zufallszahl = Anfang + ((Ende-Anfang) * RND)

Für ganze Zufallszahlen verwenden Sie

Zufallszahl = INT(Anfang + ((Ende-Anfang+1) * RND))

Kapitel: 2.5

RSET

Syntax: **RSET Übergabestring=String**

RSET ist neben LSET der zweite Befehl, der bei relativer Dateiverwaltung Daten in den Datensatzpuffer schreibt. Sie geben eine Übergabevariable an, die Sie im FIELD-Befehl definiert haben, und weisen ihr einen String zu. Dieser String wird dann in den Puffer übertragen. Das R bei RSET steht für "Rechtsbündig." Das bedeutet: Wenn die Daten kürzer sind als das Feld, das im Puffer dafür reserviert ist, werden die Daten rechtsbündig in das Datenfeld übertragen. Das heißt, die Leerstellen stehen links, am Anfang des Felds.

Das ist besonders nützlich bei Zahlenwerten, die in Rechnungen oder sonst irgendwie formatiert ausgedruckt werden sollen. Um einen Datensatzpuffer einzurichten, benötigen Sie den FIELD-Befehl. Und um die Daten vom Puffer auf Diskette zu bringen, brauchen Sie PUT. Übrigens, RSET und LSET lassen sich auch

außerhalb der relativen Dateiverwaltung einsetzen. Sie bringen Strings linksbündig oder rechtsbündig in andere Strings. Probieren Sie mal:

```
Form$=SPACE$(20)
RSET Form$="Amiga"
WRITE Form$
```

RUN

Syntax: RUN [Label]
RUN (Dateiname) [,R]

Der RUN-Befehl startet das aktuelle BASIC-Programm. Vorher wird der Bildschirm gelöscht, das LIST-Window wird in den Hintergrund gelegt, und alle offenen Dateien werden geschlossen. Sie können RUN sowohl im BASIC-Window als auch im Programm benutzen. Wenn Sie hinter RUN ein Label angeben, startet das Programm mit dem angegebenen Programmteil. Geben Sie hinter RUN einen Dateinamen an, wird das Programm geladen und gestartet. Hängt hinten noch ein ,R dran, bleiben alle offenen Dateien weiterhin geöffnet. So können Sie ein Programm nachladen, das mit den bisherigen Dateien weiterarbeitet.

Kapitel: 1.4

SADD

Syntax: Wert=SADD(String)

Diese Funktion ist besonders für Maschinensprache-Programmierer interessant. Sie liefert die Adresse, ab der der angegebene String im Speicher steht. Wenn Sie neue Strings im Programm verwenden, müssen Sie die Startadressen neu abfragen, weil AmigaBASIC die Strings öfter im Speicher hin und herschiebt.

SAVE

Syntax: **SAVE [(Dateiname)] [,A] [,B] [,P]**

Mit SAVE speichern Sie ein Programm auf Diskette, Festplatte oder der RAM-Disk ab. Wenn Sie keinen Dateinamen angeben, bittet Sie AmigaBASIC in einem Dialog-Window, den Namen für das Programm einzugeben. Wenn Sie hinter dem Dateinamen ,A verwenden, wird das Programm als ASCII-Datei abgespeichert. Sie brauchen diese Dateiform, wenn Sie Programme mit MERGE verbinden wollen oder wenn Sie ein AmigaBASIC-Programm in einer Textverarbeitung verändern wollen.

Mit ,B wird das Programm "binär" gespeichert. Das ist die normale Speicherungsform. Sie können ,B also auch weglassen. Die Befehle werden in Form von *Token* abgespeichert. Mit ,P wird das Programm in einer verschlüsselten, geschützten ("protected") Form abgespeichert. Es kann geladen und gestartet werden, Sie können es aber nicht mehr listen oder verändern. Heben Sie auf jeden Fall eine unverschlüsselte Version auf, falls sich mal Änderungen ergeben sollten!

Kapitel: 1.8, Zwischenspiel 3

SAY

Syntax: **SAY Phonem-String [,Stimm-Datenfeld%]**

Der SAY-Befehl spricht einen String, der aus Phonem-Codes besteht. Um normalen Text in Phonem-Codes umzuwanden, gibt es die BASIC-Funktion TRANSLATE$. Sie können den Text natürlich auch selbst in Phonem-Schreibweise angeben. Die Regeln dafür sind in Anhang H des Commodore-AmigaBASIC-Handbuchs aufgeführt. Hinter dem Phonem-String können Sie den Namen eines Integerfelds angeben, das mindestens 9 Elemente beinhaltet. Diese Elemente beeinflussen die Stimme, die der Amiga zum Sprechen verwendet:

Feld%(0) gibt die Frequenz der Sprache an. Dadurch wird die Stimmhöhe eingestellt. Möglich sind Werte zwischen 65 (tief) und 320 (hoch), Grundeinstellung (also der Wert, der verwendet wird, wenn Sie kein Datenfeld angeben) ist 110.

Feld%(1) gibt an, ob betont (0) oder computerhaft monoton (1) gesprochen werden soll. Grundeinstellung ist 0.

Feld%(2) legt die Sprechgeschwindigkeit fest. Diesen Wert geben Sie in Worten pro Minute an. Möglich sind Werte zwischen 40 (langsam) und 400 (schnell), Grundeinstellung ist 150.

Feld%(3) bestimmt, ob die Sprache männlich (0) oder weiblich (1) klingen soll.

Feld%(4) legt die Samplingfrequenz fest. Dieser Wert beeinflußt akustisch vor allem die Stimmhöhe. Mögliche Werte liegen zwischen 5000 (tief) und 28000 (hoch), Grundeinstellung ist 22200. Extreme Werte sind hier schon nicht mehr verständlich.

Feld%(5) bestimmt die Lautstärke. Die erlaubten Werte liegen zwischen 0 (nichts zu hören) und 65 (laut). Grundeinstellung ist 65.

Feld%(6) legt den Kanal fest, auf dem die Sprache ausgegeben wird. Bitte vergleichen Sie Tabelle 10 im Kapitel 6.3. Grundeinstellung ist 10 (= jedes freie Paar aus rechtem und linkem Kanal).

Feld%(7) legt fest, ob AmigaBASIC das Programm während der Sprachausgabe anhalten (0) oder die Sprache im Hintergrund laufen soll, während das Programm weiterarbeitet (1).

Feld%(8) wählt den Modus, falls Sie in Feld%(7) eine 1 angeben: Was soll passieren, wenn zwei SAY-Befehle aufeinander folgen? 0 bedeutet: Erst wird der alte SAY-Befehl beendet, dann beginnt

der neue. 2 bedeutet: Der neue Befehl wird sofort ausgeführt, die alte Sprachausgabe wird abgebrochen. 1 bedeutet (unabhängig vom Wert in Feld%(7)): Der aktuelle Sprachbefehl wird nicht ausgeführt. Solange dieser Wert auf 1 steht, kann kein SAY stattfinden.

Kapitel: 6.2

SCREEN

Syntax: **SCREEN Nummer,Breite,Höhe,Bitebenen,Auflösungsstufe**

Mit dem SCREEN-Befehl erzeugen Sie einen neuen Screen. Unter AmigaBASIC können neben dem Workbench-Screen noch vier weitere Screens verwendet werden (Nummern 1 bis 4). Die 'Breite' und die 'Höhe' geben die Ausmaße des Screens in Pixels an, nicht aber die Auflösungsstufe. Die Werte sollten kleiner oder gleich der gewählten Ausflösungstufe sein.

An vierter Stelle geben Sie die Anzahl der Bitebenen an. Dieser Wert entscheidet über die Anzahl der möglichen Farben auf dem Screen. Bei einer Bitebene können Sie 2 Farben verwenden, bei 2 Bitebenen 4 Farben, bei 3 Bitebenen 8 Farben, bei 4 Bitebenen 16 Farben und bei 5 Ebenen 32 Farben. Die 'Auflösungsstufe' ist eine Zahl zwischen 1 und 4.

Standard-Auflösung:

Stufe	Beschreibung	Speicherbedarf pro Bitebene
1	320*200 Punkte	8000 Bytes
2	640*200 Punkte	16000 Bytes
3	320*400 Punkte (Interlace)	16000 Bytes
4	640*400 Punkte (Interlace)	32000 Bytes

PAL-Auflösung:

Stufe	Beschreibung	Speicherbedarf pro Bitebene
1	320*256 Punkte	10240 Bytes
2	640*256 Punkte	20480 Bytes
3	320*512 Punkte (Interlace)	20480 Bytes
4	640*512 Punkte (Interlace)	40960 Bytes

Kapitel: 2.3

SCREEN CLOSE

Syntax: SCREEN CLOSE Nummer

Da ein Screen ziemlich viel Speicherplatz benötigt, sollten Sie ihn abschalten, wenn Sie ihn nicht mehr benötigen. Dazu gibt es den Befehl SCREEN CLOSE. Er nimmt den Screen vom Schirm, egal ob sich Windows darauf befinden oder nicht, und gibt den benutzten Speicherbereich wieder frei.

SCROLL

Syntax: SCROLL (x1,y1)-(x2,y2), x-Richtung, y-Richtung

Mit diesem Befehl können Sie einen Bildschirmausschnitt in eine beliebige Richtung verschieben. (x1,y1)-(x2,y2) legt ein Rechteck fest. Der Inhalt dieses Rechtecks wird verschoben. 'x-Richtung' legt fest, um wieviele Punkte der Inhalt in horizontaler Richtung verschoben werden soll. Positive Werte verschieben den Bildausschnitt nach rechts, negative nach links. 'y-Richtung' macht dasselbe für die vertikale Richtung. Positive Werte veschieben den Ausschnitt nach unten, negative Werte nach oben. Die Bildteile, die aus dem Bereich herausgeschoben werden, werden gelöscht.

Kapitel: 1.17

SGN

Syntax: `Wert=SGN(Wert)`

SGN ist die Vorzeichenfunktion: Sie liefert:

-1	für negative Zahlen	('Wert'<0)
0	für die Zahl 0	('Wert'=0)
1	für positive Zahlen	('Wert'>0)

Mit dieser Funktion können Sie in Programmen das Vorzeichen einer Variablen feststellen.

SHARED

Syntax: `SHARED Variable [Variable, ...]`

Dieser Befehl gibt innerhalb eines SUB-Programms an, welche Variablen des Hauptprogramms auch im SUB-Programm verwendet werden sollen. Geben Sie hinter SHARED die Variablennamen an. Datenfelder kennzeichnen Sie durch ein leeres Paar Klammern:

```
SUB Test STATIC
SHARED Testnummer, Testfeld(), Test$
```

Sie können mehrere SHARED-Befehle angeben, aber nur innerhalb von SUB-Programmen. Vergleichen Sie auch den DIM-Befehl, er bietet nämlich eine eigene SHARED-Option.

Kapitel: 4.5

SIN

Syntax: **Wert=SIN(Wert)**

Die allseits beliebte und geschätze Sinusfunktion. SIN liefert den Sinus eines Winkels im Bogenmaß. Wie Sie einen Gradmaßwinkel ins Bogenmaß umrechnen können, finden Sie bei COS.

Kapitel: 2.5

SLEEP

Syntax: -

Dieser Befehl ist im Rahmen von Event Trapping interessant. Er hält das Programm an, bis eines der Ereignisse eintritt, die durch Event Trapping überprüft werden können. Dazu ist es aber nicht nötig, daß Event Trapping für das betreffende Ereignis aktiviert ist. Folgende Ereignisse beenden SLEEP:

- Drücken einer Taste auf der Tastatur
- Klicken mit der linken Maustaste
- Auswahl eines Menüpunkts in einem Pulldown
- Kollision von Grafikobjekten
- ein TIMER-Ablauf

Praktisch ist SLEEP zum Beispiel auch, wenn Sie den Anwender auffordern, "irgendeine" Taste zu drücken. Im Gegensatz zu INKEY$ erkennt SLEEP nämlich auch Tasten wie <SHIFT>, <ALT>, <CTRL> oder die <Amiga>-Tasten.

SOUND

Syntax: **SOUND Frequenz, Dauer [,Lautstärke] [,Kanal]**

Dieser Befehl erzeugt einen Ton. Sie können die Frequenz in Hertz angeben, möglich sind 20 Hertz bis 15000 Hertz. Tabelle 12 in Kapitel 7.2 zeigt Ihnen, welche Frequenzen welchen musikalischen Noten entsprechen. Die Dauer des Tons ist ein Wert zwischen 0 und 77. Wenn Sie einen Ton von einer Sekunde Dauer spielen wollen, müssen Sie ungefähr 18 angeben. Bitte vergleichen Sie Tabelle 13 im Kapitel 7.2. Optional können Sie eine Lautstärke zwischen 0 und 255 angeben. Fehlt dieser Wert, verwendet AmigaBASIC die goldene Mitte: 127 Auch eine Kanalzuordnung des Tons ist möglich. Hier stehen vier Werte zur Auswahl:

0 und 3 - linker Kanal
1 und 2 - rechter Kanal

Voreinstellung (auch für BEEPs) ist Kanal 0.

Kapitel: 7.2

SOUND RESUME
SOUND WAIT

Syntax: -

Um mehrere Töne gleichzeitig oder eine Melodie im Hintergrund zu spielen, legen Sie die Töne in einer Warteschlange ab. SOUND WAIT bewirkt, daß alle nachfolgenden SOUND-Befehle nicht sofort ausgeführt werden sondern gespeichert bleiben. Denken Sie bei mehrstimmiger Musik daran, daß auch Kanäle, die vorübergehend keinen Ton spielen, besetzt werden müssen. Am besten tun Sie das mit einem Ton in 0-Lautstärke.

SOUND RESUME startet die Wiedergabe der Ton-Schlange.

Kapitel: 7.3

SPACE$

Syntax: **String=SPACE$(Länge)**

Diese Stringfunktion erzeugt einen String aus lauter Leerstellen in der angegebenen Länge. 'Länge' muß zwischen 0 und 32767 liegen. Solche Strings sind manchmal zur Formatierung von Bildschirm- und Druckerausgaben hilfreich. Oder auch zum Löschen einzelner Bildschirmzeilen.

Kapitel: 5.2

SPC

Syntax: **PRINT SPC(x);**

Dieser Befehl erzeugt in einem PRINT-Befehl die angegebene Anzahl ('x') an Leerstellen. 'x' muß zwischen 0 und 255 liegen. SPC ist oft beim Aufbau von Bildschirmmasken hilfreich.

SQR

Syntax: **Wert=SQR(Wert)**

SQR errechnet die Quadratwurzel (engl.: SQuare Root) einer Zahl.

SQR(4) ist 2, SQR(2) ist 1.414214

STATIC

siehe SUB...STATIC

STICK

Syntax: Wert=STICK(x)

Mit dem STICK-Befehl können Sie Joysticks abfragen, die am Amiga angeschlossen sind. Der Wert 'x' in Klammern hinter dem STICK-Befehl gibt an, welche Information Sie haben wollen:

Wert:	Information:
0	x-Bewegung von Joystick 1
1	y-Bewegung von Joystick 1
2	x-Bewegung von Joystick 2
3	y-Bewegung von Joystick 2

STICK(x) liefert 1, wenn der abgefragte Joystick in der angegebenen Richtung nach oben bzw. nach rechts gedrückt wird.

STICK(x) liefert 0, wenn am abgefragten Joystick in der angegebenen Richtung keine Bewegung erfolgt.

Und STICK(x) liefert -1, wenn die Bewegung nach unten bzw. links stattfindet. Vorsicht mit Joystick 1! Wenn an diesem Anschluß kein Joystick, sondern die Maus hängt, blockiert die Maus, sobald Sie STICK(0) oder STICK(1) verwenden!

Kapitel: 3.5

STOP

Syntax: -

Dieser Befehl bricht das laufende BASIC-Programm ab. Durch den CONT-Befehl können Sie das Programm fortsetzen. STOP schließt keine offenen Dateien. Das sollten Sie bei der Rückkehr in den Direktmodus bedenken.

STR$

Syntax: `String=STR$(Wert)`

Die Funktion STR$ wandelt eine Zahl in einen String um. Dabei werden aber keine Bytes oder etwas Ähnliches erzeugt, sondern einfach die Zahl mit allen ihren Ziffern Stelle für Stelle in einen String gesteckt.

STR$(4095) ist "4095".

Kapitel: 3.4

STRIG

Syntax: `Wert=STRIG(x)`

Mit STRIG fragen Sie ab, ob der Feuerknopf an einem Joystick gedrückt wird. Der Wert 'x' in Klammern legt fest, welche Information Sie haben wollen:

STRIG(0) fragt ab, ob der Feuerknopf von Joystick 1 seit dem letzten STRIG(0) gedrückt wurde. Wenn ja, ist das Ergebnis -1, wenn nein, ist das Ergebnis 0.

STRIG(1) fragt ab, ob der Feuerknopf von Joystick 1 gerade gedrückt wird. -1 = ja, 0 = nein

STRIG(2) fragt ab, ob der Feuerknopf von Joystick 2 seit dem letzten STRIG(2) gedrückt wurde. -1 = ja, 0 = nein

STRIG(3) fragt ab, ob der Feuerknopf von Joystick 2 gerade gedrückt wird. -1 = ja, 0 = nein

Weiteres bei STICK.

Kapitel: 3.5

STRINGS$

Syntax: String=STRING$(Länge,ASCII-Code)
String=STRING$(Länge,String)

STRINGS$ erzeugt einen String in der angegebenen Länge, der aus lauter gleichen Zeichen besteht. Sie können für das Zeichen entweder den ASCII-Code angeben,

```
PRINT STRING$(100,191)
```

oder einen String, dessen erstes Zeichen benutzt wird:

```
PRINT STRING$(100,"A")
```

Kapitel: 1.16

SUB...STATIC

Syntax: SUB Name [(Variable, ...)] STATIC

Mit diesem Befehl beginnt ein SUB-Programm. SUB-Programme sind Unterprogramme, die unabhängig vom restlichen Programm sind. Sie verwenden eigene Variablen, können aber mit dem SHARED-Befehl auf Variablen des Hauptprogramms zugreifen. Jedes SUB-Programm hat einen Namen, mit dem es aufgerufen werden kann. Dazu benutzen Sie den CALL-Befehl, der aber in eindeutigen Fällen auch entfallen kann. Der Name darf bis zu 40 Zeichen lang sein. Sie können dem Programm Variablen übergeben. Dabei müssen Sie darauf achten, daß diese vom Variablentyp zu den Angaben in der SUB...STATIC-Zeile passen. Datenfelder kennzeichnen Sie wieder durch ein Klammernpaar:

```
SUB Hallo (Test, Farben(), Alpha) STATIC
```

Ein SUB-Programm wird mit dem Befehl END SUB beendet. Die Programmteile zwischen SUB und END SUB werden nur

nach Aufruf ausgeführt. Steht ein SUB-Programm mitten im Hauptprogramm, springt AmigaBASIC darüber weg. Der Befehl EXIT SUB ermöglicht Ihnen, ein SUB-Programm schon vor dem END SUB-Befehl zu verlassen.

Kapitel: 4.5

SWAP

Syntax: **SWAP Variable1, Variable2**

Dieser Befehl dient dazu, zwei Variablen gegeneinander auszutauschen. Nach der Ausführung hat 'Variable1' den ehemaligen Wert von 'Variable2' und umgekehrt. Die beiden Variablen müssen vom Variablentyp her übereinstimmen.

SYSTEM

Syntax: -

Dieser Befehl beendet AmigaBASIC. Der Amiga kehrt auf die Workbench oder ins CLI zurück, je nachdem, wie Sie AmigaBASIC aufgerufen haben. Wenn das aktuelle Programm noch nicht abgespeichert wurde, macht Sie AmigaBASIC in einem Dialog-Window darauf aufmerksam und bietet Ihnen die Möglichkeit, das Programm doch noch abzuspeichern.

Kapitel: 1.9

TAB

Syntax: **PRINT TAB(x);**

Auch diesen Befehl benutzen Sie, um Druckausgaben zu formatieren: AmigaBASIC füllt den Raum von der aktuellen Druckposition zur Spalte 'x', die hinter TAB in Klammern angegeben ist, mit Leerzeichen auf. Die Druckausgabe beginnt also in der

Spalte 'x'. Befindet sich die aktuelle Druckposition hinter Spalte 'x', wird in der nächsten Zeile ab Spalte 'x' gedruckt. TAB können Sie nur mit den Befehlen PRINT, PRINT# und LPRINT verwenden.

TAN

Syntax: **Wert=TAN(Wert)**

Neben SIN und COS gibt es noch eine dritte trigonometrische Funktion. (Für Mathematiker: Sie wissen schon. Für Nicht-Mathematiker: Tri-go-no-me-tri-sche Funktionen sind ein Teil der Dinge, mit denen Lehrer ihre Schüler zu ärgern pflegen.) TAN liefert den Tangens eines Winkels im Bogenmaß. Wie Sie Grad-Angaben ins Bogenmaß umrechnen, ist bei COS erklärt.

TIME$

Syntax: **String=TIME$**

TIME$ liefert die Uhrzeit, von der Ihr Amiga zur Zeit ausgeht. Diese Uhrzeit können Sie in Preferences einstellen, sie ist nicht automatisch richtig. Sie erhalten einen 8 Zeichen langen String, der so aufgebaut ist: SS:MM:SS, also Stunden:Minuten:Sekunden.

Kapitel: 1.17

TIMER

Syntax: **Wert=TIMER**

TIMER mal als Funktion: Sie erhalten die Uhrzeit in Sekunden. Das heißt, die Anzahl der Sekunden, die seit Mitternacht (00:00:00) vergangen sind. Dabei geht der Amiga natürlich wie-

der von seiner inneren Uhr aus, die nicht unbedingt stimmen muß und die Sie mit Preferences stellen können bzw. die über eine eingebaute Echtzeituhr beim Starten des Amiga gestellt wurde..

Kapitel: 1.17

TIMER ON
TIMER OFF
TIMER STOP

Syntax: -

Diese Befehle steuern das Timer-Event-Trapping. TIMER ON aktiviert es, TIMER OFF schaltet es ab. Mit TIMER STOP werden alle Timer-Events bis zum nächsten TIMER ON unterdrückt. Mit dem Befehl ON TIMER GOSUB geben Sie an, welches Unterprogramm die Timer-Unterbrechungen bearbeitet.

TRANSLATE$

Syntax: Phonem-String=TRANSLATE$(String)

Diese Stringfunktion übersetzt einen String in Phonem-Code. Das dient als Vorbereitung für den SAY-Befehl. Dabei geht TRANSLATE$ in der gegenwärtigen Version von AmigaBASIC von amerikanischen Aussprache- und Betonungsreglen aus. Wenn Sie deutschen Text sprechen wollen, müssen Sie entweder TRANSLATE$ austricksen ("Fishers Frits fisht frisha Fisha" - Sie erinnern sich.) oder aus Phonemen selbst einen Phonem-String zusammensetzen. Lesen Sie auch die Erklärungen zum SAY-Befehl.

Kapitel: 6.2

TRON
TROFF

Syntax: -

Diese beiden Befehle steuern die Trace-Funktion. TRON schaltet Trace an, TROFF schaltet es aus. Bei aktivierter Trace-Funktion wird der aktuelle Befehl im LIST-Window von einem orangen Kästchen umrahmt. Das ist beim Austesten von Programmen sehr hilfreich.

Kapitel: 1.14

UBOUND

siehe LBOUND

UCASE$

Syntax: **String=UCASE$(String)**

UCASE$ wandelt einen angegebenen String in Großbuchstaben um.

    UCASE$("Amiga") ist "AMIGA"

Besonders beim alphabetischen Sortieren von Texten und beim Abfragen von Benutzereingaben ist diese Funktion hilfreich.

Kapitel: 1.16

VAL

Syntax: **Wert=VAL(String)**

VAL ist die Gegenfunktion zu STR$: Ein String, der eine ausgeschriebene Zahl beinhaltet, wird in einen Wert umgewandelt. Dabei werden so lange die Zeichen des Strings benutzt und umgerechnet, bis der erste Buchstabe auftaucht:

VAL("1234Hallo") ergibt 1234
VAL("1234Hallo4567") ergibt auch 1234.
VAL("Hallo1234") ergibt 0.

Kapitel: 1.7

VARPTR

Syntax: **Wert=VARPTR(Variable)**

Ähnlich wie SADD bei Strings liefert VARPTR die Startadresse, ab der eine angegebene Variable im Speicher steht. VARPTR brauchen Sie besonders häufig, wenn Sie Maschinensprache-Unterprogramme von AmigaBASIC aus benutzen wollen. Lesen Sie dazu auch den CALL-Befehl.

WAVE

Syntax: **WAVE Kanalnummer, Integerfeld%**
WAVE Kanalnummer, SIN

Mit WAVE können Sie die Wellenform verändern, die vom SOUND-Befehl verwendet wird. Entweder Sie geben SIN an, dann wird eine Sinusschwingung erzeugt. Das ist auch die Grundeinstellung, wenn Sie WAVE überhaupt nicht benutzen. Oder Sie geben ein Integerfeld an, das mindestens 256 Elemente beinhaltet. In diesem Integerfeld ist der Verlauf einer beliebigen

Wellenform gespeichert. Die Werte liegen zwischen -128 und +127 und geben die Amplitude der Welle zum Zeitpunkt des Samplings an. Falls Sie jetzt nur noch Bahnhof verstehen, lesen Sie bitte Kapitel 7.4.

Die angegebene Wellenform wird dann einem oder mehreren der vier Tonkanäle (0 und 3 - links, 1 und 2 - rechts) zugewiesen. Nach dieser Zuweisung sollten Sie das Feld mit ERASE löschen, um Speicherplatz zu sparen.

Kapitel: 7.5

WHILE...WEND

Syntax:
```
WHILE (Bedingung)
 [(Befehle)]
WEND
```

WHILE...WEND führt eine Schleife aus, solange eine angegebene Bedingung zutrifft. Wenn die Bedingung nicht mehr zutrifft, macht das Programm hinter WEND weiter. Ein typisches Beispiel ist das Warten auf einen Tastendruck:

```
WHILE INKEY$="" : WEND
```

Sie können WHILE...WEND-Schleifen auch beliebig ineinander verschachteln.

Kapitel: 2.8

WIDTH

Syntax:
WIDTH [Breite] [,Tabulatorabstand]
WIDTH [LPRINT] [,Breite] [,Tabulatorabstand]
WIDTH #Dateinummer [,Breite] [,Tabulatorabstand]
WIDTH Gerätebezeichnung [,Breite] [,Tabulatorabstand]

Sie sehen, hinter WIDTH steckt mehr, als man zunächst vermutet. Es gibt vier verschiedene Syntax-Versionen, die teilweise auch mit Datenverwaltung zu tun haben.

WIDTH dient grundsätzlich dazu, die Zeilenbreite für eine Ausgabe festzulegen. Da der Amiga aus dem rechten Bildschirmrand hinausschreibt und keinen automatischen Zeilenvorschub erzeugt, müssen Sie in Ihren BASIC-Programmen die Bildschirmbreite festlegen. Das geht mit der ersten angegebenen Syntax:

> Bei 60 Zeichen pro Zeile verwenden Sie WIDTH 61.
> Bei 80 Zeichen pro Zeile muß es WIDTH 76 heißen.

Der zweite Wert hinter WIDTH gibt die Abstände zwischen zwei Tabulatorpositionen an.

Die beiden Werte für Zeilenbreite und Tabulatorabstände können Sie auch für den Drucker angeben. Und zwar entweder mit der Option LPRINT hinter WIDTH, oder mit der dritten Syntax-Version: Öffnen Sie einfach eine Druckerdatei und geben Sie die Dateinummer an. Die mit WIDTH festgelegten Werte beziehen sich aber nur auf die angegebene Datei. Druckausgaben mit LPRINT sind nicht betroffen.

Nach der vierten Syntax können Sie die Werte auch direkt für ein Gerät angeben (SCRN, LPT1, COM1). Die Zeilenbreite und Tabulatorabstände werden dann nach dem nächsten OPEN einer Datei zu dem angegebenen Gerät verwendet. Ist bereits eine Datei geöffnet, ist sie noch nicht betroffen. Voreinstellung für den Drucker sind 80 Zeichen pro Zeile.

WINDOW

Syntax: **WINDOW Windownummer [,Titel] [,(x1,y1)-(x2,y2)] [,Windowtyp] [,Screen]**

Mit diesem Befehl können Sie von AmigaBASIC aus neue Windows erzeugen und benutzen. Jedes Window hat eine Nummer. Sie ermöglicht Ihnen, das Window anzusprechen oder auszuwählen. Das BASIC-Window hat die Nummer 1, die vom Benutzer erzeugten Windows haben die Nummern 2 und höher. Bei der Erzeugung eines Windows können Sie einen Titel und die Größe des Windows angeben. Sie legen dazu den linken oberen und den rechten unteren Eckpunkt als Bildschirmkoordinaten (bezogen auf den verwendeten Screen) fest. Wenn Sie keine Größenangabe machen, nimmt AmigaBASIC die volle Größe des gewählten Screens an. 'Windowtyp' ist eine Summe aus Einzelwerten, die bestimmte Eigenschaften und Möglichkeiten des Windows beeinflussen:

1 Sie können das Window mit dem Größensymbol verkleinern oder vergrößern.

2 Sie können das Window mit der Maus an der Bewegungsleiste festhalten und bewegen.

4 Das Window besitzt zwei Hintergrund-/Vordergrund-Symbole, mit denen Sie es nach hinten oder nach vorne klicken können.

8 Das Window besitzt ein Schließsymbol, mit dem es ausgeklickt werde kann.

16 Der Windowinhalt ist bei Bewegungen oder Überlagerungen durch andere Windows gespeichert und wird wieder aufgebaut. Ist dieser Wert nicht gewählt, verliert das Window bei Bewegungen oder Überlagerungen seinen Inhalt. Die letzte Option muß in jedem Fall benutzt werden, wenn Sie im Window PAINT-Befehle verwenden wollen.

Optional können Sie noch den Screen angeben, auf dem das Window erscheinen soll. Wenn Sie alle Angaben weglassen und den WINDOW-Befehl nur folgendermaßen verwenden,

WINDOW Windownummer

wird das angegebene Window in den Vordergrund geholt und zum Ausgabewindow gemacht.

Kapitel: 2.4

WINDOW

Syntax: Wert=WINDOW(x)

WINDOW gibt es auch als BASIC-Funktion. Sie können damit Daten über das aktuelle Ausgabewindow erfahren. 'x' darf zwischen 0 und 8 liegen und gibt an, welche Information Sie wünschen.

WINDOW(0) liefert die Nummer des aktivierten Windows. Das ist nicht immer gleichzeitig das Ausgabewindow, sondern das Window, in das der Anwender mit der Maus zuletzt geklickt hat. (Sie erkennen dieses Window daran, daß sein Titel nicht in Geisterschrift gedruckt ist.)

WINDOW(1) liefert die Nummer des Ausgabewindows, also des Windows, in das AmigaBASIC alle Bildschirmausgaben schickt.

WINDOW(2) sagt Ihnen die Breite des aktuellen Ausgabewindows in Pixels.

WINDOW(3) sagt Ihnen die Höhe des aktuellen Ausgabewindows in Pixels.

WINDOW(4) teilt Ihnen in Pixels die x-Koordinate mit, an der im aktuellen Ausgabewindow das nächste Zeichen erscheinen wird. Sie erfahren also, wo zur Zeit der Ausgabecursor steht.

WINDOW(5) verrät die y-Koordinate der nächsten Bildschirmausgabe.

WINDOW(6) sagt Ihnen, wieviele Farben im aktuellen Ausgabewindow erlaubt sind. Sie erhalten die höchste zulässige Farbnummer.

WINDOW(7) gibt die Adresse an, ab der Intuition die Daten des aktuellen Ausgabewindows speichert. Dieser Datenbereich ist die sogenannte INTUITION-WINDOW-Struktur.

WINDOW(8) ergibt ebenfalls die Adresse eines Intuition-Datenbereichs, nämlich der sogenannten RASTPORT-Struktur. Und zwar der RASTPORT-Struktur für das aktuelle Ausgabewindow. Die letzten beiden Werte nutzen Ihnen allerdings nur, wenn Sie genaue Kenntnisse über das Amiga-Betriebssystem haben.

Kapitel: 2.4, 2.9, 4.3

WINDOW CLOSE

Syntax: WINDOW CLOSE Windownummer

Dieser Befehl nimmt das angegebene Window vom Bildschirm. Es wird dabei aber nicht gelöscht! Sie können weiterhin Ausgaben in das Window schicken, obwohl es gar nicht mehr zu sehen ist.

Kapitel: 2.4

WINDOW OUTPUT

Syntax: WINDOW OUTPUT Windownummer

Das angegebene Fenster wird zum aktuellen Ausgabefenster gemacht, ohne daß es dabei in den Vordergrund geholt wird. Das ist der einzige Unterschied zu der bekannten Variante

WINDOW Windownummer.

So können Sie auch Ausgaben in teilweise verdeckte oder nicht sichtbare Windows schicken.

Kapitel: 2.4

WRITE

Syntax: **WRITE [Variable oder Wert] [Trennzeichen] [Variable oder Wert]...**

WRITE gibt wie PRINT ab der aktuellen Druckposition Zeichen auf dem Bildschirm aus. Als Trennzeichen sind nur Komma (,) oder Strichpunkt (;) erlaubt. Die Unterschiede zu PRINT: WRITE gibt zwischen den Einzelwerten grundsätzlich Kommas aus, und Strings werden automatisch von Anführungszeichen umgeben. Deshalb ist WRITE besonders geeignet zur Ausgabe von Werten in Dateien.

WRITE#

Syntax: **WRITE# Dateinummer, [Variable oder Wert] [Trennzeichen] [Variable oder Wert...]**

Funktioniert wie WRITE, schreibt aber seine Ausgaben in eine Datei. Dieser Befehl ist besonders praktisch, um Strings in sequentielle Dateien zu schreiben, da sie automatisch in Anführungszeichen eingebettet werden. Beim Lesen erhalten Sie so immer den String in voller Länge, auch wenn er Trennzeichen beinhaltet. Nur Anführungszeichen dürfen im String natürlich nicht vorkommen!

Kapitel: 3.4

XOR

Syntax: **Wert = Wert1 XOR Wert2**

Die XOR-Verknüpfung verknüpft die einzelnen Bits von 'Wert1' und 'Wert2' folgendermaßen:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Kapitel: Zwischenspiel 4

Anhang C: Die Programme aus der "BASICDemos"-Schublade

Zusammen mit AmigaBASIC liefert Commodore auf der "ExtrasD"-Diskette freundlicherweise einige Beispielprogramme. Sie befinden sich in der Schublade "BASICDemos". Das Programm "Bitte-lesen" gibt Hinweise darüber, was die Programme tun und wie sie funktionieren. Aber sicher möchten Sie nicht immer erst ein Programm laden, um die anderen zu benutzen. Deshalb gibt es diesen Anhang, in dem wir Ihnen die einzelnen Programme kurz vorstellen.

Bitte-lesen

Wie schon erwähnt, ist das eine Art Anleitung für die anderen Programme. Das Programm besteht nur aus Kommentarzeilen. Alle beginnen mit einem Hochkomma ('). Im LIST-Window können Sie die Texte lesen. Das Programm ist für die Schrift-Einstellung 80 Zeichen pro Zeile gedacht, nur so können Sie den kompletten Text im LIST-Window problemlos lesen. Wenn Sie wollen, können Sie das Programm auf einem Drucker ausdrucken, dann haben Sie auch diese Informationen schriftlich.

Music

In diesem Programm macht AmigaBASIC Musik. Das Ergebnis ist wirklich toll: Die Leute, die das Programm geschrieben haben, haben sich viel Mühe gegeben, und eine dreistimmige Melodie in DATA-Zeilen abgelegt. Um das Ganze grafisch zu untermalen, entsteht auch noch eine Liniengrafik.

Der Programmteil 'PlaySong:' liest die DATA-Werte aus und errechnet Frequenzen daraus. In den DATA-Zeilen stehen die Noten als Buchstaben (c, d, e, f, g...), außerdem Informationen über die Länge einer Note, einen Oktavenwechsel und Pausen.

Für die Wellenform wird ein besonderer Trick verwendet: Anstatt die Werte für die gewünschte Sinusschwingung direkt ausrechnen zu lassen, wurden die berechneten Werte in DATA-Zeilen abgelegt, weil Einlesen schneller geht als Ausrechnen. Sie wissen ja: AmigaBASIC und ein schlechter Mathe-Schüler haben viel gemeinsam. Die Formel, nach der die Werte entstanden, steht vor den DATAs im Listing.

Das Zeichnen der Grafik wird durch Event Trapping gelöst: Alle zwei Sekunden läuft der Timer ab, und ON TIMER ruft das Unterprogramm 'TimeSlice:' auf. Die Größe der Grafik paßt sich automatisch der aktuellen Windowgröße an.

Library

Hier wird vorgeführt, wie man mit LIBRARY-Funktionen zum Beispiel verschiedene Schriftarten erzeugen kann. Das funktioniert jedoch nur mit den beiden Schriften "Topaz 60" und "Topaz 80", da nur diese beiden Schriftarten im ROM-*Kernel* stehen. Alle anderen Schriften finden Sie ja im Verzeichnis "Fonts:" auf der Workbench-Diskette. Für das Programm werden die beiden Libraries "graphics.bmap" und "dos.bmap" benutzt. Wie man das macht, lesen Sie bitte bei den Befehlen DECLARE FUNCTION...LIBRARY und LIBRARY im Anhang B nach.

Das SUB-Programm 'DosLibDemo' zeigt, wie man AmigaDOS-Funktionen von AmigaBASIC aus aufrufen kann. Um diesen Teil benutzen zu können, muß AmigaBASIC aber vom CLI gestartet worden sein, nicht von der Workbench.

lib2

Dieses Programm basiert auf dem "Library"-Programm. Es wurde jedoch so verändert, daß auch beliebige Schriften von Diskette geladen werden können. (Bei "Library" geht es ja nur mit den beiden ROM-Zeichensätzen...)

BitPlanes
ScreenPrint

Durch Library-Programmierung wird vieles möglich, was normalerweise in AmigaBASIC nicht geht. Diese beiden Programme sind ein gutes Beispiel dafür. "BitPlanes" durchsucht die Listen, die wir im Kapitel 4.3 kennengelernt haben. (Sogenannte C-Strukturen). Dadurch wird es dann möglich, die Startadressen von Screens und Windows, oder auch die aktuelle Farbbelegung auszulesen.

"ScreenPrint" ist eine Hardcopy-Routine, die sehr aufwendig und unter Library-Einsatz in AmigaBASIC realisiert wurde. Wer sich wirklich mal ausführlich in die Verwendung von Libaries hineingraben möchte, ist mit diesem Programm sicher gut beraten. Das Programm "BitPlanes" können Sie durch einen Mausklick beenden, "ScreenPrint" stoppt, sobald Sie die <Q>-Taste drücken.

Screen

Auf einem 320 * 200-Punkte-Screen mit 32 Farben (5 Bitebenen) werden mit AREA farbige Vierecke gezeichnet. Die Lage der Eckpunkte berechnet das Programm durch SIN- und COS-Formeln. Durch PATTERN kommen auch noch einige einfache Füllmuster ins Spiel. Wenn Sie das Window vergrößern oder verkleinern, verändert sich die Größe der Vierecke entspechend. Die Farben werden auf Zufallsbasis festgelegt.

Demo

Dieses Programm erzeugt vier Windows und zeigt ein bißchen Multitasking in BASIC: Im ersten Window springen die beiden Bälle, im zweiten Window bewegen sich Linien, ähnlich wie im "Music"-Programm. Im dritten Window erscheinen die Vierecke, die wir aus "Screen" kennen, und im vierten Window werden verschieden große Kreise farbig ausgemalt. Das erste Window können Sie vergrößern. Sobald die Bälle gemerkt haben, daß das

Window größer wurde, nutzen sie auch den neuen Platz. Die anderen drei Windows können durch Ausklicken abgeschaltet werden, dann werden die anderen entsprechend schneller. Das Window mit den Bällen (es trägt den Namen "Animation") entpuppt sich als verkleinertes BASIC-Window. Die SUB-Programme 'NextLine', 'NextPoly' und 'NextCircle' werden innerhalb einer WHILE...WEND-Schleife nacheinander aufgerufen. Sie sorgen für die Aktionen in den Windows 2 bis 4, während die Object-Animation in Window 1 durch Event Trapping gesteuert ist. Die Kollision eines Balls mit dem anderen oder dem Rahmen ist diesmal das auslösende Ereignis. Das Aussehen der Bälle liest "Demo" übrigens aus der Datei "ball".

CLogo

In diesem Programm rührt Commodore kräftig die Werbetrommel für das eigene Firmenzeichen. Auf Zufallsbasis werden mit Grafik-Befehlen verschieden große Versionen des Commodore-C gezeichnet.

Kaleidoskop

Dieses Programm zeigt einige tolle Farbgrafikeffekte. Es ist den Kaleidoskop-Programmen nachempfunden, die es mittlerweile schon von einigen Anbietern gibt. Auf Zufallsbasis werden Flächengrafiken erzeugt. Die Anzahl der verwendeten Bit-Ebenen kann am Anfang des Programms zwischen 3 und 5 gewählt werden. Wenn Sie das Programm beenden wollen, drücken Sie einfach die <Q>-Taste..

Picture/Picture2

Das sind zwei Kopien desselben Programms. Im BASIC-Handbuch von Commodore wird dieses Programm verwendet, um eine erste Bekanntschaft mit AmigaBASIC zu schließen. Dabei setzt Commodore gleich Maussteuerung und Grafik-GET und -

PUT ein: Mit der Maus können Sie die CIRCLE-Grafik über den Bildschirm steuern. Da Sie das Programm in einem Kapitel des Handbuchs verändern sollen, gibt es zwei Versionen auf Diskette.

ObjEdit

Den kennen Sie schon: Es ist der ObjectEditor, mit dem Sie die Grafikobjekte für Object-Animation erzeugen können. Wie Sie dieses Programm bedienen, steht ausführlich in Kapitel 1.11.

Nur ein Hinweis: Wenn Sie 512K RAM besitzen und Sie das Flimmern bei Objects nicht stört, können Sie im Listing des Object-Editors die Anzahl der erlaubten Farben erhöhen: Zu Beginn steht hinter Hochkommas der Aufbau einer Object-Datei, dann kommt ein DEF FN und ein DIM-Befehl. Im dann folgenden Programmteil sehen Sie am Ende vier Zeilen, die mit einem Hochkomma (') geschützt sind. Wenn Sie die Hochkommas entfernen, können Sie in der Variablen 'Depth' die Anzahl der Bitebenen angeben. Zur Auswahl stehen 3 und 4, da das Programm nur auf einem 640 * 200-Punkte-Screen arbeitet. Aber 16 Farben sind ja auch schon eine ganze Menge. Bedenken Sie, daß in der Object-Datei keine Farben abgespeichert werden. Sie müssen für die richtigen Farben also im einlesenden Programm sorgen.

Und denken Sie daran, daß jede zusätzliche Bitebene den Flimmereffekt verstärkt, der in der aktuellen Version von AmigaBASIC bei Bob-Animationen auftritt. Um das "ObjEdit"-Programm genau zu verstehen, braucht man schon einige Erfahrungen in AmigaBASIC. Wenn Sie mal Zeit und Lust haben, sollten Sie sich den Aufbau des ObjectEditors genauer anschauen. Das Programm ist wie unser Malprogramm vollständig mit Event Trapping gelöst.

Kommunik

Dieses Programm zeigt, wie man mit der seriellen Schnittstelle umgeht. Wenn dort eine Datenleitung mit 9600 Baud Übertragungsrate angeschlossen ist, können Sie Zeichen senden und empfangen. Ohne so einen Anschluß bietet das Programm natürlich nicht viel Interessantes.

ConvertFd

Das ist ein Hilfsprogramm für Leute, die viel mit LIBRARY-Routinen arbeiten: Die LIBRARY-Dateien müssen für die Benutzung in AmigaBASIC in einem speziellen Format auf Diskette stehen, dem sogenannten .BMAP-Format. Vier Libraries dieser Art, nämlich "exec.bmap", "diskfont.bmap", "dos.bmap" und "graphics.bmap" gibt es ja in der "BASICDemos"-Schublade, sie werden von den Programmen "Library", "lib2" und einigen anderen benutzt. Aus allen Libraries, deren Name mit .FD endet, können Sie mit "ConvertFd" eine .BMAP-Datei erzeugen. Solche FD-Dateien finden Sie in der Schublade "FD1.2" auf Ihrer "Extras"-Diskette. Solange Sie sich mit Libraries noch nicht auskennen, sollten Sie "ConvertFD" allerdings nicht benutzen.

Speech

Dieses Programm funktioniert ähnlich wie unser Sprach-Utility. Zuerst verlangt AmigaBASIC die Workbench. Wenn Sie nur ein Laufwerk besitzen, legen Sie also bitte die Workbench-Diskette ein. Danach können Sie dann eine Textzeile eingeben, die nach Drücken der <RETURN>-Taste gesprochen wird. Es gibt sechs Schieberegler, die die Stimme beeinflussen: "Pitch" steht für die Stimmhöhe, also die Frequenz. "Inflection" hat nur zwei mögliche Stellungen und regelt die Betonung (menschlich moduliert oder monoton). "Rate" verändert die Sprechgeschwindigkeit, und "Voice" die Stimme (auch hier gibt es nur zwei mögliche Regler-

positionen: Männlich und weiblich). "Tune" steht für die Samplingfrequenz, verändert also auch die Stimmlage, und "Volume" schließlich regelt die Lautstärke. Das Programm ist nur zum Ausprobieren gedacht - Werte abspeichern kann man damit nicht.

Speechd

Das kleine d macht's: Commodore Deutschland hat das "speech"-Programm um eine Routine erweitert, die das Sprechen deutscher Sprache ermöglicht. Die Regler haben jetzt deutsche Beschriftungen, ganz unten ist ein Schalter für "Deut/Engl" (also Deutsch oder Englisch) dazugekommen. Die deutsche Tastatur wird unterstützt. Zum Beenden des Programms geben Sie das Wort "Ende" ein.

Ein amerikanischer Akzent ist aber nicht zu verleugnen. Das ist auch klar, denn die Routine 'initdeutsch:' muß versuchen, die deutschen Laute möglichst gut mit den englischen Phonemen zu beschreiben. Über die verschiedenen Möglichkeiten, den Amiga deutschen Text sprechen zu lassen, erfahren Sie mehr in Kapitel 6.2.

SuchZeich

Dieses kleine Utility ist bei der Arbeit mit Dateien recht nützlich: Es sucht eine angegebene Zeichenkette in einer angegebenen Datei und zeigt, an welcher Stelle die gesuchten Zeichen gefunden wurden. Die gesuchten Werte können Sie dezimal, hexadezimal oder als String angeben.

Vergl

Dieses Programm funktioniert ähnlich wie "Suchzeich". Der einzige Unterschied ist, daß hier zwei Dateien miteinander verglichen werden. Die Zeichen, durch die sich die beiden Dateien unterscheiden, werden vom Programm auf dem Bildschirm angezeigt.

Dreier

Bei diesem Programm handelt es sich um ein sehr hübsches Grafik-Demo. Allein durch Dreiecksflächen erzeugt der AREA-Befehl farbenprächtige Gebilde. Diese Gebilde "bewegt" das Programm von Zeit zu Zeit noch zusätzlich durch eine Farbpalettenanimation. Immer mal wieder wird auch die gesamte Farbpalette geändert.
Am Programmanfang fragt Sie das Programm, ob es auf einem PAL- oder einem NTSC-Amiga läuft. Von der Eingabe (p oder n) hängt ab, wie groß das verwendete Window wird. Der Programmautor regt Sie übrigens selbst im Listing an, mit der Farbweiterschaltung zu spielen. In den Formeln für 'fs' und 'ff' können Sie festlegen, wann die Farbpalette durchwechselt und wann sie geändert wird.

Zum Beenden des Programms drücken Sie <Q>.

LoadACBM
LoadILBM-SaveACBM
SaveILBM

Hier sind Sie nun also, die Utilities, die Commodore auf die "ExtrasD"-Diskette gepackt hat, um das Problem der fehlenden Lade- und Speicherbefehle für IFF-Grafiken in AmigaBASIC auszugleichen. Auch in diesen drei Programmen spielen Libraries die Hauptrolle, ohne sie ginge nämlich überhaupt nichts. Als erstes werden Sie sich wahrscheinlich fragen, was denn bitteschön

"ACBM" bedeuten soll. Dabei handelt es sich um einen von Commodore definierten neuen IFF-Typ. Näheres zu den Vor- und Nachteilen von ACBMs können Sie im Zwischenspiel 5 lesen.

ACBMs können in BASIC schneller gelesen werden als ILBMs, die normalen IFF-Grafik-Dateien. Doch was hilft der Geschwindigkeitsgewinn, wenn ein Programm dafür keine "normalen" IFF-Bilder mehr versteht? Denn Programme wie DPaint oder GrafiCraft speichern ja nach wie vor ILBMs ab. So blieb Commodore nichts anderes übrig, als insgesamt drei Programme bereitzustellen:

"LoadILBM-SaveACBM" liest normale ILBMs ein, rechnet sie um und speichert sie als ACBMs ab. Diese ACBMs können dann mit dem Programm "LoadACBM" eingelesen werden. Und "SaveILBM" macht das Trio komplett, dieses Programm erzeugt eine bewegte Liniengrafik und speichert sie als ILBM ab. Ein wichtiger Vorteil der drei Routinen: Sie sind auch in der Lage, Farbpalettenanimationen darzustellen. Die Daten, die dazu benötigt werden, befinden sich gegebenenfalls in einem Chunk namens "CCRT" innerhalb der IFF-Datei.

Beachten Sie bitte: Falls Sie diese drei Routinen in eigene BASIC-Programme einbinden wollen, sorgen Sie bitte dafür, daß am Ende der Kette eine ILBM-Datei herauskommt. Wir wollen ja schließlich kompatibel bleiben...

Anhang D: Das kleine Fachwortlexikon

Die *kursiv* gedruckten Fachwörter konnten wir oft noch nicht bei ihrem ersten Auftreten besprechen, weil das im Text zu weit führen würde. In diesem Anhang finden Sie alle Erklärungen. Wir haben noch ein paar Begriffe zusätzlich aufgenommen, so können Sie dieses kleine Fachwortlexikon auch zum Nachschlagen verwenden.

AmigaDOS

DOS ist eine Abkürzung für "Disk Operating System". Das ist das Disketten-*Betriebssystem* des Amiga. Dieses Programm ist für den Datenaustausch zwischen Amiga und Diskettenlaufwerken zuständig und kümmert sich außerdem um die Organisation der Daten auf Diskette. Sie können AmigaDOS direkt Befehle geben, und zwar über CLI, den 'Command Line Interpreter' auf der Workbench.

ASCII-Code

Engl.: American Standard Code für Information Interchange. Deutsch: Amerikanischer Standardcode für Informationsaustausch. Dieser genormte Code weist jedem Zeichen einen bestimmen Byte-Wert zu. Eine ASCII-Tabelle finden Sie im Anhang B beim CHR$-Befehl.

BASIC

Engl.: Beginners all purpose symbolic instruction code. Deutsch: Symbolische Programmiersprache für Anfänger. Jetzt wissen Sie also auch, woher BASIC seinen Namen hat. Er stimmt aber eigentlich nicht so ganz. BASIC ist weder eine reine Anfänger-Sprache noch besonders stark symbol-gesteuert. Aber als die ersten BASIC-Versionen entwickelt wurden, standen ihnen so komplizierte Sprachen wie Assembler, COBOL und FORTRAN gegenüber. So erklärt sich aus damaliger Sicht der Name.

Betriebssystem

Das Betriebssystem ist das Programm, das ständig in einem Computer läuft, um Grundfunktionen wie Tastatureingaben, Bildschirmdarstellung, Betrieb der Schnittstellen usw. zu ermöglichen. Beim Amiga setzt sich das Betriebssystem aus mehreren Teilen zusammen. Vergleichen Sie auch *Intuition*, *Kernel* und *AmigaDOS*.

Bildschirm-Editor

Ein *Editor* ist für die Steuerung des *Cursors* etc. zuständig. Bei einem Bildschirm-Editor kann sich der *Cursor* auf dem ganzen Bildschirm frei bewegen und an jeder beliebigen Stelle Eingaben oder Korrekturen vornehmen. Im LIST-Window von Amiga-BASIC wird ein Bildschirm-Editor verwendet. Im Gegensatz dazu steht ein *Zeilen-Editor*.

Binärzahlen

Auch: Dualzahlen oder Zweiersystem. Das ist ein Zahlensystem mit der Basis 2. Bei jeder Zweierpotenz wird eine neue Stelle eröffnet. Als Ziffern verwendet man meistens 0 und 1. Die Zahlen 0 bis 10 sehen binär so aus:

0	0000	6	0110
1	0001	7	0111
2	0010	8	1000
3	0011	9	1001
4	0100	10	1010
5	0101		

Näheres darüber erfahren Sie im Zwischenspiel 4.

Bit

Engl.: binary digit. Ein Bit ist die kleinste Einheit bei der Informationsspeicherung. Es kann entweder "an" oder "aus" sein, entspricht "gesetzt" oder "gelöscht", entspricht "wahr" oder "falsch", entspricht 0 oder 1. Acht Bits zusammen bilden ein *Byte.*

Bit-Ebene

Bei der Speicherung von Grafiken entspricht jeder Bildpunkt einem Bit. Damit mehrere Farben dargestellt werden können, kommen pro Bildpunkt (*Pixel*) weitere Bits dazu. Die Grafiken bestehen so aus verschiedenen Bit-Ebenen. Die "übereinander" liegenden Bits bestimmen die Farbe eines Bildpunkts. Bild 5 im Kapitel 2.2 zeigt, wie Sie sich Bit-Ebenen am besten vorstellen können.

Byte

Jede Speicherzelle im Amiga speichert genau ein Byte. Ein Byte besteht aus 8 Bits. Diese Einteilung stammt aus der Zeit der 8-Bit-Prozessoren. Ein Byte kann einen Wert zwischen 0 (binär 00000000) und 255 (binär 11111111) speichern. Meistens wird zur Speicherung eines Zeichens oder eines Buchstabens ein Byte verwendet. Weil ein KByte (ein Kilobyte) 2^{10} Bytes beinhaltet, sind 1 KByte nicht 1000 Bytes, sondern 1024 Bytes. 256K entsprechen 262144 Bytes. Und 512K sind 524288 Bytes. Sie bekommen also sogar ein paar Bytes mehr als gedacht.

Cursor

So heißt die Marke, die Ihnen auf dem Bildschirm anzeigt, wo Sie gerade schreiben oder arbeiten. Beim Amiga gibt es verschiedene Cursor-Arten. Da wäre zunächst der Mauscursor, der kleine Pfeil, der sich synchron zur Maus bewegt. Dann gibt es

den BASIC-Cursor, der im LIST-Window und im BASIC-Window anzeigt, wo die Tastatureingaben stattfinden. Andere Programme benutzen manchmal auch noch andere Cursor-Arten.

D/A-Wandler

Digital/Analog-Wandler. Sie wandeln ein digitales Signal, also eine Reihe von Zahlenwerten, in einen analogen Strom. Je größer die Zahl, desto stärker das Stromsignal. Auf diese Weise funktioniert zum Beispiel die Musikerzeugung beim Amiga. Der Chip "Paula", der die Musik macht, hat vier D/A-Wandler eingebaut. Um Musik zu digitalisieren, also in Zahlenwerte umzuwandeln, bräuchten Sie das Gegenteil: einen A/D-Wandler. Er funktioniert genau andersrum.

Dezimalzahlen

Das ist das normale, allen vertraute Zehnersystem. Es basiert auf der Zahl 10 und hat zehn Ziffern: 0, 1, 2, 3, 4, 5, 6, 7, 8 und 9. Für Menschen das Natürlichste der Welt, nur Computer tun sich damit etwas schwer. Lesen Sie mal das Zwischenspiel 4.

Directory

Das englische Wort für Inhaltsverzeichnis (auch Telefonbuch). Mit Directory meint man bei Computern das Inhaltsverzeichnis einer Diskette.

Drucker

Ein *Peripheriegerät* zum Ausdrucken von Daten, Text oder Grafiken auf Papier. Sie können verschiedene Drucker am Amiga anschließen, die Anpassung führen Sie mit dem Programm Preferences auf der Workbench-Diskette durch. Die meisten Drucker werden an der Parallel-Schnittstelle des Amiga angeschlossen, einige auch an der seriellen Schnittstelle. Näheres dazu

können Sie im Kapitel 3.5 nachlesen. Es gibt verschiedene Arten von Druckern: Matrixdrucker (die häufigsten), Tintenstrahldrucker (die leisesten), Laserdrucker (die besten und die teuersten), Typenraddrucker (die schönsten - gleich nach den Laserdruckern). Die Unterscheidung richtet sich nach dem Prinzip, wie die Zeichen aufs Papier kommen. Typenraddrucker können keine Grafiken drucken, haben dafür aber das Schriftbild einer Schreibmaschine. Durch *Near Letter Quality* ist der Qualitätsabstand von Matrixdruckern zu Typenraddruckern aber sehr klein geworden.

Editor

Ein Editor ist der Teil eines Programms oder eines *Betriebssystems*, der für die *Cursor*-Steuerung und damit für das Eingeben, Einfügen, Löschen und Korrigieren von Texten auf dem Bildschirm zuständig ist. Vergleichen Sie *Bildschirm-Editor* und *Zeilen-Editor*.

Hardcopy

Das ist ein Bildschirmausdruck auf einem *Drucker*. Meist spricht man von Hardcopies, wenn's um Grafiken geht. Aber auch Textausdrucke des Bildschirminhalts nennt man Hardcopy. AmigaBASIC tut sich mit Hardcopies etwas schwer, Sie brauchen ein Hilfsprogramm dazu, zum Beispiel "GraphicDump" aus der "System"-Schublade der Workbench.

Hardware

So nennt man die Geräte und Bauteile eines Computers. Hardware kann man anfassen, oder: (sehr beliebte, weil anschauliche Definition) Hardware geht kaputt, wenn man sie fallenläßt. Im Gegensatz zur *Hardware* steht die *Software*.

Hexadezimalzahlen

Auch ein sehr beliebtes Zahlensystem bei Computern. Das Hexadezimalsystem basiert auf der Zahl 16. Es hat die Ziffern 0, 1, 2, 3, ... 8, 9, A, B, C, D, E und F. Man kennzeichnet Hexadezimalzahlen gern durch ein vorangestelltes $-Zeichen. So sehen die ersten 35 Hexadezimalzahlen aus:

0	$0	9	$9	18	$12	27	$1B
1	$1	10	$A	19	$13	28	$1C
2	$2	11	$B	20	$14	29	$1D
3	$3	12	$C	21	$15	30	$1E
4	$4	13	$D	22	$16	31	$1F
5	$5	14	$E	23	$17	32	$20
6	$6	15	$F	24	$18	33	$21
7	$7	16	$10	25	$19	34	$22
8	$8	17	$11	26	$1A	35	$23

Für weitere Informationen lesen Sie bitte das Zwischenspiel 4.

IFF

Interchange File Format. Von der Firma Electronic Arts entwickeltes Speicherformat für Amiga-Daten. Lesen Sie dazu bitte Kapitel 4.2.

Interlace

Zwischenzeilenmodus. Durch eine Verdopplung der dargestellten Bildzeilen auf dem Monitor erreicht der Amiga eine höhere Auflösung. Lesen Sie bitte Kapitel 2.2.

Interpreter

Deutsch: Übersetzer. Nichts anderes ist AmigaBASIC nämlich. Der Amiga versteht BASIC nicht direkt. Das ist auch kein Wunder, denn er denkt ja nur in "Strom an, Strom aus". Der BASIC-Interpreter übersetzt die BASIC-Befehle in für den Amiga und seinen *Prozessor* verständliche Maschinensprache-Befehle.

Intuition

Das ist ein weiterer Teil des Amiga-*Betriebssystems*. Intuition ist für die Arbeit mit Windows etc. zuständig. Viele Routinen aus Intuition können von anderen Programmen als Unterprogramme benutzt werden.

Joystick

Engl.: Steuerknüppel. Zwei von diesen Spielgeräten können Sie am Amiga anschließen.

Kernel

Übersetzt: Kerngehäuse. Denken Sie bitte nur bedingt an einen Apfel. Kernel nennt man den Teil des Betriebssystems, der sozusagen Basis für alles andere ist. Das sind die Input/Output-Routinen etc. Das Kernel wird beim Amiga 1000 von der Kickstart-Diskette geladen und kann danach im Speicher nicht mehr verändert werden. Bei Amiga 500 und 2000 befindet sich das Kernel im *ROM*.

kompatibel, Kompatibilität

Bedeutet "austauschbar", kann zusammenarbeiten. Sind zwei Geräte softwarekomptibel, können sie dieselbe Software verwenden. Der Amiga kann teilweise zum IBM PC kompatibel gemacht werden. (Dafür gibt es eine Softwarelösung, den "Transformer",

und eine Hardwarelösung namens "Sidecar" bzw. die PC-Karte im Amiga 2000.) Sind zwei Programme datenkompatibel, können sie gemeinsam dieselben Daten verwenden. Beim Amiga ist das recht häufig der Fall, dank *IFF*.

Maschinensprache

Das ist die Sprache, die der 68000-*Prozessor* des Amiga direkt versteht. Sie besteht letztlich nur aus Nullen und Einsen, kann aber schrittweise für den Menschen verständlicher gemacht werden. Wer z.B. in Maschinensprache programmiert, verwendet Kürzel wie MOVEA, MOVEC, CMPI oder SBCD. Jetzt wissen Sie auch, warum Maschinensprache-Progammierer BASIC-Befehle furchtbar einfach finden.

Multitasking

Die Fähigkeit, mehrere Programme unabhängig voneinander gleichzeitig auszuführen. Während der Anwender mit einem Programm arbeitet, laufen die anderen im Hintergrund weiter. So kann man Zeit sparen und mehrere Aufgaben gleichzeitig lösen lassen. Da der *Prozessor* seine Arbeitskapazität auf die einzelnen Tasks (Aufgaben) verteilen muß, wird ein einzelnes Programm langsamer, wenn mehrere Programme gleichzeitig laufen.

Near Letter Quality

Deutsch: Fast-Schreibmaschinen-Qualität. Matrixdrucker haben oft eine schwer lesbare, unschöne Punktschrift. Durch verschiedene Tricks kann man aber mit ihnen eine Schrift erzeugen, die der von Schreibmaschinen oder Typenraddruckern nur noch wenig oder überhaupt nicht mehr nachsteht. Für Laserdrucker ist NLQ (so die Abkürzung) gleich gar keine Schwierigkeit mehr.

Peripheriegerät

Ein Zusatzgerät, das Sie an den Amiga anschließen können. Also z.B. *Drucker*, Plotter, Zusatz-Laufwerke, Festplatten-Laufwerke, Grafiktableaus, Scanner, Joysticks etc.

Pixel

Ein Bildpunkt in einer Grafik. Beim Amiga wird durch das Prinzip der *Bit-Ebenen* ein Pixel durch bis zu fünf Bits im Speicher vertreten. Ein Computerbild setzt sich aus Pixels zusammen wie ein Zeitungsfoto aus Rasterpunkten. Je mehr Pixels auf den Bildschirm passen, umso höher ist die Auflösung, umso realistischer sieht das entstehende Bild aus. Allerdings spielen die Farben dabei auch noch eine wichtige Rolle. Mehr darüber im Kapitel 2.2

Prozessor

Das "Gehirn" des Computers. Dieser Chip steuert die wichtigsten Funktionen. Beim Amiga heißt er 68000. (Prozessoren haben immer so phantasievolle Namen.) Ein Prozessor kann ungeheuer schnell einzelne Befehlsschritte in *Maschinensprache* ausführen. Der 68000 schafft fast 8 Millionen dieser Kleinst-Arbeitsschritte pro Sekunde. Und damit er sich dabei auf das wirklich Wichtige konzentrieren kann, werden ihm viele Routineaufgaben wie Datenaustausch, Bildschirmdarstellung etc. von seinen Co-Prozessoren Agnus, Denise und Paula abgenommen.

RAM

Engl.: Random Access Memory. Deutsch: Schreib-/Lesespeicher. In diesen Speicher können Sie eigene Werte schreiben und später wieder auslesen. Einziges Problem: Sein Inhalt wird beim Ausschalten des Amiga gelöscht. Deshalb müssen Daten, die im RAM stehen, auf Diskette oder Festplatte abgespeichert werden.

ROM

Engl.: Read Only Memory. Deutsch: Nur-Lese-Speicher, Festwertspeicher. Im Gegensatz zum *RAM* wird das ROM beim Ausschalten nicht gelöscht. Dafür können Sie ins ROM keine eigenen Daten schreiben. Was dort einmal steht, ist so leicht nicht mehr wegzubekommen. Für *Betriebssysteme* etc. ist das auch sehr gut so. Im Amiga 500 und 2000 befindet sich der Inhalt der ehemaligen Kickstart-Diskette in einem ROM.

Beim Amiga 1000 gibt es eigentlich gar kein richtiges ROM. Zumindest nicht sehr viel davon. Nur das Ladeprogramm für die Kickstart-Diskette (das auch die schöne Hand mit ihrer Diskette darstellt), befindet sich in ROMs. Der RAM-Bereich, wo das *Betriebssystem* steht, kann elektronisch so abgeschlossen werden, daß er sich wie ROM verhält. Beim Ausschalten wird er aber trotzdem gelöscht.

Schnittstelle

Ein Anschluß, an dem Sie *Peripheriegeräte* oder eine Datenübertragungs-Leitung anschließen können. Der Amiga hat mehrere Schnittstellen. Zum Beispiel eine Parallel-Schnittstelle, eine serielle Schnittstelle, eine Schnittstelle für zusätzliche Disketten-Laufwerke und verschiedene Schnittstellen für Monitor und Fernseher.

Screen

Übersetzt: Bildschirm. Einen Screen beim Amiga kann man an der Kopfleiste mit dem Mauscursor festhalten und nach oben oder nach unten ziehen. Auf dem Screen befinden sich dann die Windows, die Grafiken, der Text - kurz: die gesamte Bildschirmdarstellung. Ausführlicher können Sie das im Kapitel 2.3 nachlesen.

Scrolling

Kürzel aus "Screen" (Bildschirm) und "rolling" (Rollen). Scrolling bedeutet, daß Daten an einem Rand aus dem Bildschirm verschwinden und dafür an der gegenüberliegenden Seite neue Daten erscheinen. So kann man immer einen Ausschnitt einer Gesamtdarstellung auf dem Bildschirm sehen.

Software

Die Programme und Daten im Speicher eines Computers. Software kann man nicht anfassen, höchstens die Diskette, auf der sie aufgezeichnet ist. Nur durch die Software kann die *Hardware* auch richtig ausgenutzt werden.

Token

Übersetzt: Zeichen, Merkmal. Ein Ein-Byte-Code für einen BASIC-Befehl. AmigaBASIC setzt ihn beim Abspeichern von Programmen ein, um Diskettenplatz zu sparen. Anstatt für den PRINT-Befehl die Zeichen P, R, I, N und T abzuspeichern, speichert man lieber das Token 172 ab. Wenn das Programm dann wieder eingelesen wird, weiß AmigaBASIC, daß es das Token 172 durch den Befehl PRINT ersetzen muß.

Utility

Hilfsprogramm. Ein Programm, das Ihnen bei einer bestimmten Arbeit hilft. Alles, was nützlich ist, kann man als "Utility" bezeichnen. In diesem Buch finden Sie eine ganze Menge nützlicher Sachen, also auch viele Utilities.

Zeilen-Editor

Ein Editor, der nur in einer einzigen Zeile arbeitet. Sie haben also im Gegensatz zum *Bildschirm-Editor* nicht den ganzen Bildschirm auf einmal für Ihre Eingaben zur Verfügung. Der Cursor kann nicht nach oben oder unten bewegt werden, nur nach links oder rechts. Im BASIC-Window bietet AmigaBASIC nur einen Zeilen-Editor.

Anhang E: Stichwortverzeichnis

